CPSC 330
Applied Machine Learning

Lecture 17: Introduction to natural language processing

UBC 2022-23

Instructor: Mathias Lécuyer

In [1]:
```
## Imports

import os
import re
import string
import sys
import time
from collections import Counter, defaultdict

import IPython
import nltk
import numpy as np
import numpy.random as npr
import pandas as pd
from IPython.display import HTML
from ipywidgets import interactive
from nltk.corpus import stopwords
from nltk.tokenize import sent_tokenize, word_tokenize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
```

Announcements

- Homework 7 due on the 22^{nd}.
- Please take the intructor feedback survey: https://canvas.ubc.ca/courses/30777/external_tools/6073 (https://canvas.ubc.ca/courses/30777/external_tools/6073).

Learning objectives

- Broadly explain what is natural language processing (NLP).
- Name some common NLP applications.
- Explain the general idea of a vector space model.
- Explain the difference between different word representations: term-term co-occurrence matrix representation and Word2Vec representation.
- Describe the reasons and benefits of using pre-trained embeddings.
- Load and use pre-trained word embeddings to find word similarities and analogies.
- Demonstrate biases in embeddings and learn to watch out for such biases in pre-trained embeddings.
- Use word embeddings in text classification and document clustering using `spaCy`.
- Explain the general idea of topic modeling.
- Describe the input and output of topic modeling.
- Carry out basic text preprocessing using `spaCy`.

What is Natural Language Processing (NLP)?

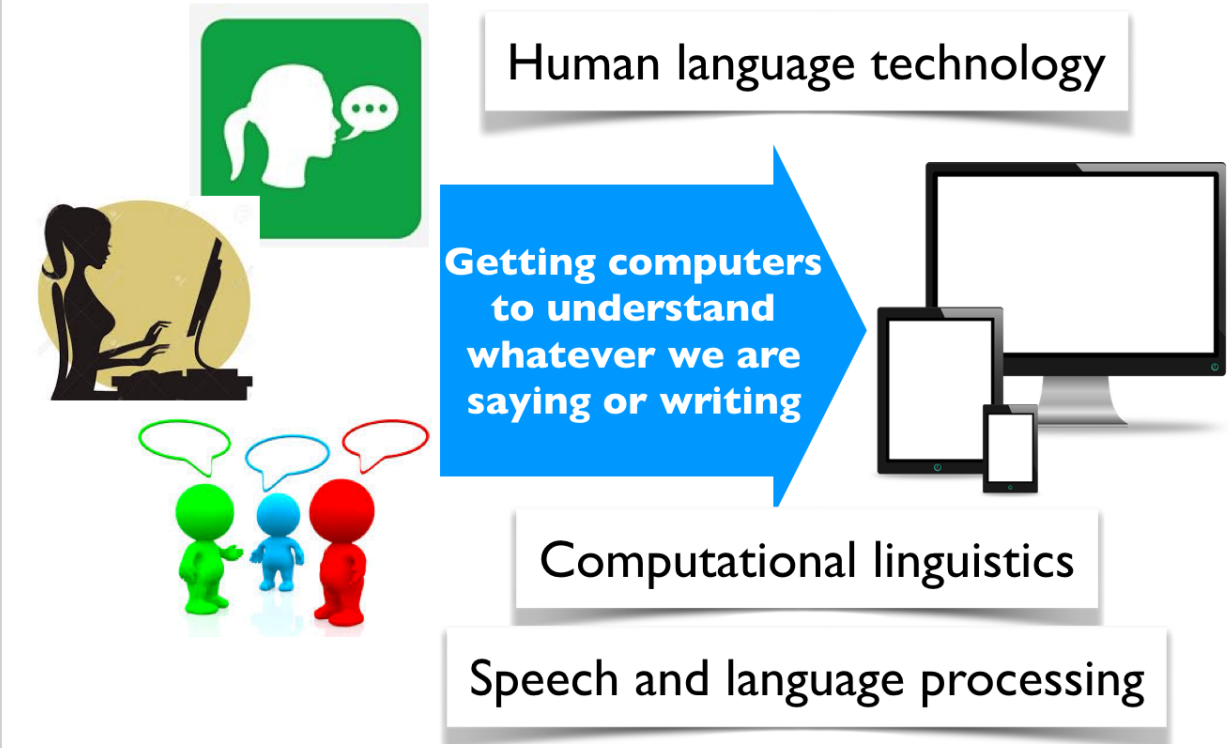
- What should a search engine return when asked the following question?

Who is Panini?

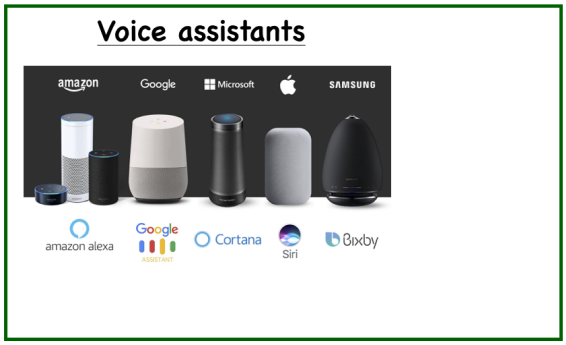
What is Natural Language Processing (NLP)?

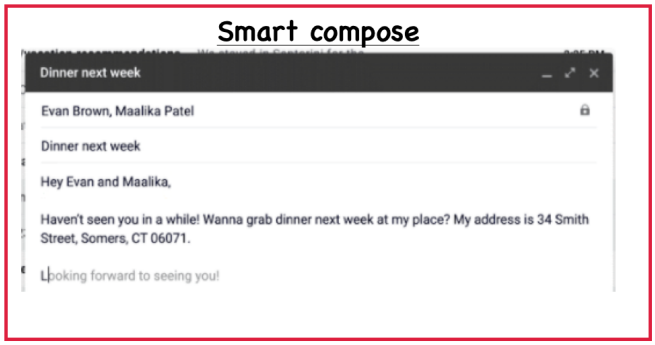
How often do you search everyday?

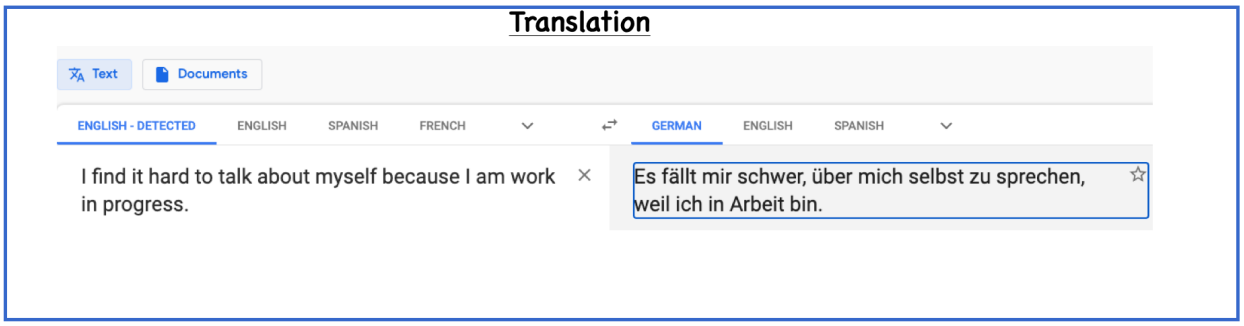
What is Natural Language Processing (NLP)?



Everyday NLP applications







NLP in news

Often you'll see NLP in news. Some examples:

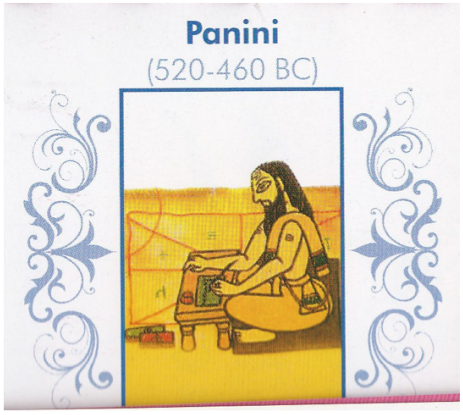
- How suicide prevention is getting a boost from artificial intelligence (https://abcnews.go.com/GMA/Wellness/suicide-prevention-boost-artificial-intelligence-exclusive/story?id=76541481)
- Meet GPT-3. It Has Learned to Code (and Blog and Argue). (https://www.nytimes.com/2020/11/24/science/artificial-intelligence-ai-gpt3.html)
- Introducing ChatGPT (https://openai.com/blog/chatgpt)
- How Do You Know a Human Wrote This? (https://www.nytimes.com/2020/07/29/opinion/gpt-3-ai-automation.html)
- ...

Why is NLP hard?

- Language is complex and subtle.
- Language is ambiguous at different levels.
- Language understanding involves common-sense knowledge and real-world reasoning.
- All the problems related to representation and reasoning in artificial intelligence arise in this domain.

Example: Lexical ambiguity

Who is Panini?



Example: Referential ambiguity

If the **baby** does not thrive on raw **milk**, boil **it**.

it = ?

[Ambiguous news headlines (http://www.fun-with-words.com/ambiguous_headlines.html)](http://www.fun-with-words.com/ambiguous_headlines.html)

> PROSTITUTES APPEAL TO POPE

- **appeal to** means make a serious or urgent request or be attractive or interesting?

> KICKING BABY CONSIDERED TO BE HEALTHY

- **kicking** is used as an adjective or a verb?

> MILK DRINKERS ARE TURNING TO POWDER

- **turning** means becoming or take up?

Overall goal

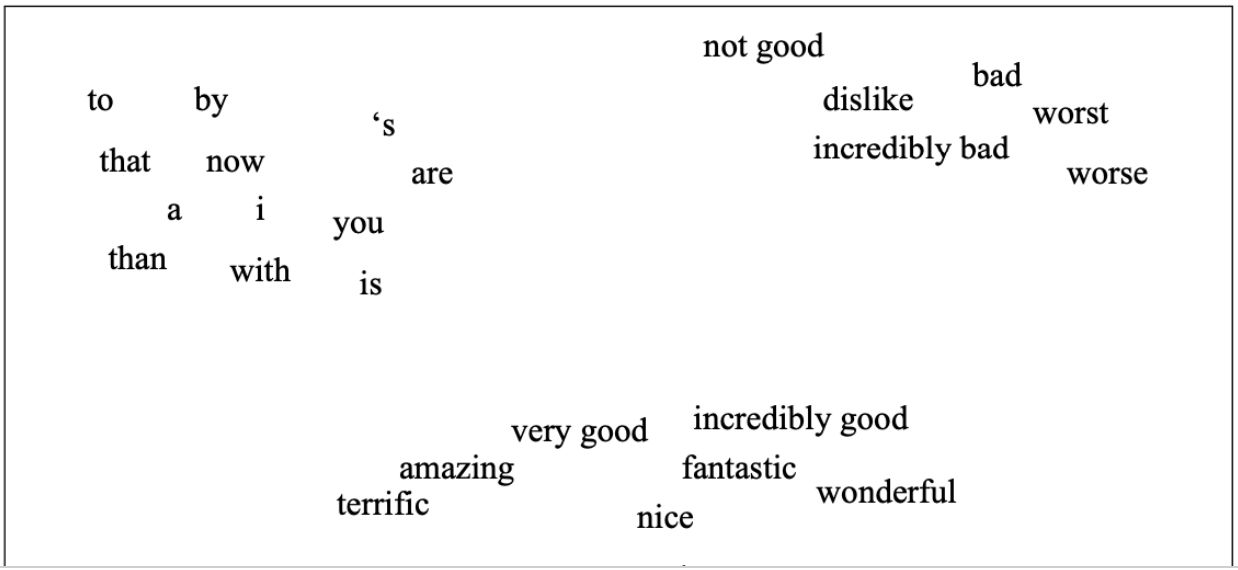
- Give you a quick introduction to you of this important field in artificial intelligence which extensively used machine learning.

Today's plan

- Word embeddings
- Topic modeling
- Basic text preprocessing

Word Embeddings

- The idea is to represent word meaning so that similar words are close together.



Why do we care about word representation?

- So far we have been talking about sentence or document representation.
- Now we are going one step back and talking about word representation.
- Although word representation cannot be directly used in text classification tasks such as sentiment analysis using tradition ML models, it's good to know about word embeddings because they are so widely used.
- They are quite useful in more advanced machine learning models such as recurrent neural networks.

Word meaning

- A favourite topic of philosophers for centuries.
- An example from legal domain: [Are hockey gloves gloves or "articles of plastics"? (https://www.scc-csc.ca/case-dossier/info/sum-som-eng.aspx?cas=36258)](https://www.scc-csc.ca/case-dossier/info/sum-som-eng.aspx?cas=36258)

> Canada (A.G.) v. Igloo Vikski Inc. was a tariff code case that made its way to the SCC (Supreme Court of Canada). The case disputed the definition of hockey gloves as either gloves or as "articles of plastics."

Word meaning: ML and NLP view

- Modeling word meaning that allows us to
  - draw useful inferences to solve meaning-related problems
  - find relationship between words,
    - E.g., which words are similar, which ones have positive or negative connotations

Word representations

How do we represent words?

- Suppose you are building a question answering system and you are given the following question and three candidate answers.
- What kind of relationship between words would we like our representation to capture in order to arrive at the correct answer?

Question: How **tall** is Machu Picchu?

Candidate 1: Machu Picchu is 13.164 degrees south of the equator.

Candidate 2: The official height of Machu Picchu is 2,430 m.

Candidate 3: Machu Picchu is 80 kilometres (50 miles) northwest of Cusco.

Need a representation that captures relationships between words.

- We will be looking at two such representations.
  1. Sparse representation with **term-term co-occurrence matrix**
  2. Dense representation with **Word2Vec**
- Both are based on two ideas: **distributional hypothesis** and **vector space model**.

Distributional hypothesis

> You shall know a word by the company it keeps.
> — Firth, 1957

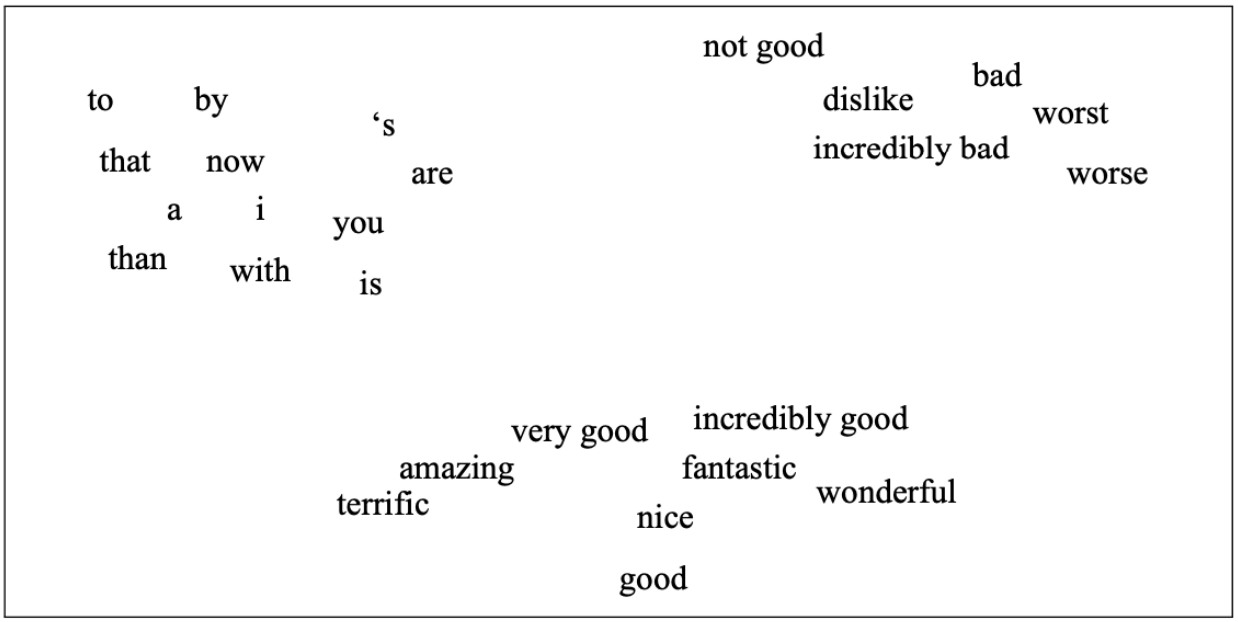
> If A and B have almost identical environments we say that they are synonyms.
> — Harris, 1954

Example:

- Her **child** loves to play in the playground.
- Her **kid** loves to play in the playground.

Vector space model

- Model the meaning of a word by placing it into a vector space.
- A standard way to represent meaning in NLP
- The idea is to create **embeddings of words** so that distances among words in the vector space indicate the relationship between them.



(Attribution: Jurafsky and Martin 3rd edition)

Term-term co-occurrence matrix

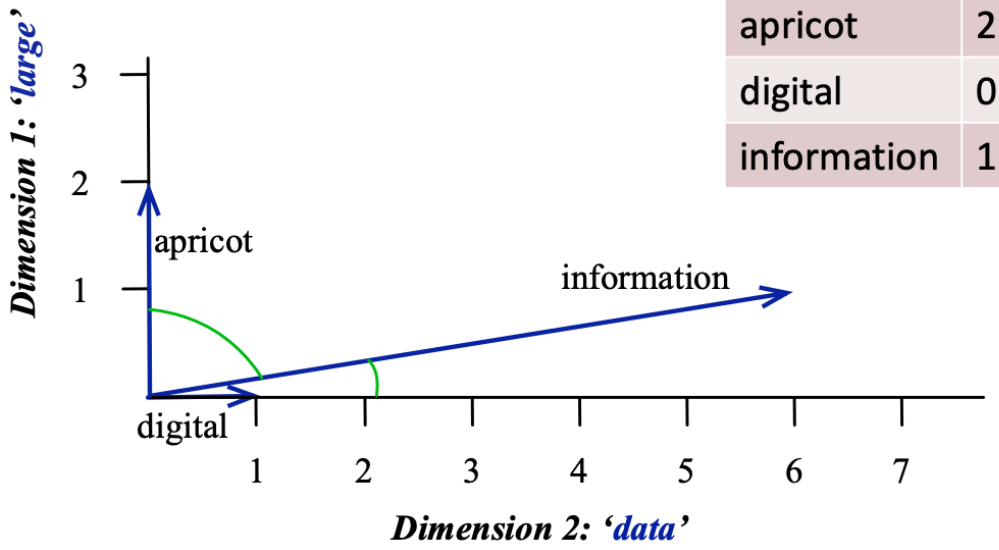
- So far we have been talking about documents and we created document-term co-occurrence matrix (e.g., bag-of-words representation of text).
- We can also do this with words. The idea is to go through a corpus of text, keeping a count of all of the words that appear in context of each word (within a window).
- An example:

	large	data
apricot	2	0
digital	0	1
information	1	6

(Credit: Jurafsky and Martin 3rd edition)

Visualizing word vectors and similarity

	large	data
apricot	2	0
digital	0	1
information	1	6

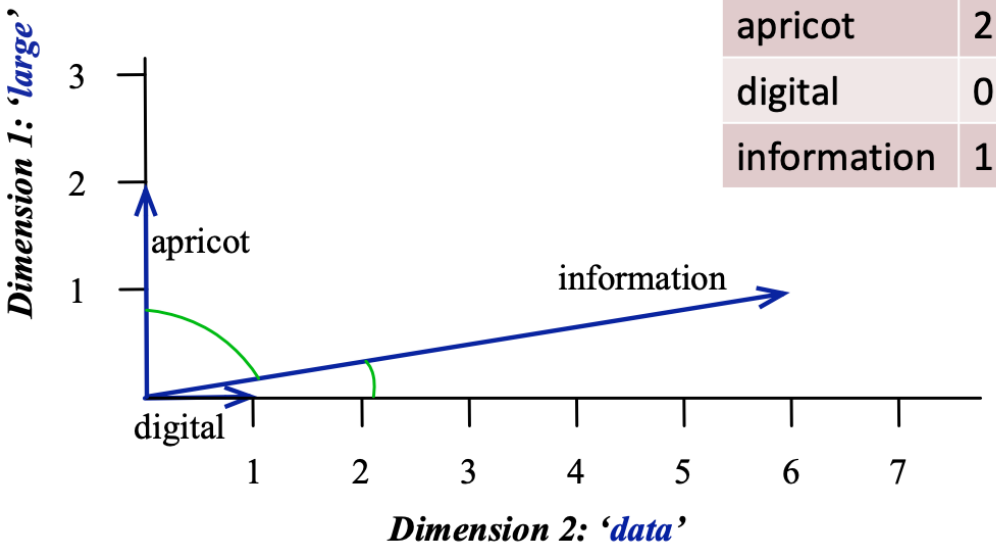


(Credit: Jurafsky and Martin 3rd edition)

- The similarity is calculated using dot products between word vectors.
  - Example: $\vec{\text{digital}}.\ \vec{\text{information}} = 0 \times 1 + 1 \times 6 = 6$
  - Higher the dot product more similar the words.

Visualizing word vectors and similarity

	large	data
apricot	2	0
digital	0	1
information	1	6



(Credit: Jurafsky and Martin 3rd edition)

- The similarity is calculated using dot products between word vectors.
  - Example: $\vec{\text{digital}}.\ \vec{\text{information}} = 0 \times 1 + 1 \times 6 = 6$
  - Higher the dot product more similar the words.
- We can also calculate a normalized version of dot products (related to the angle between word vectors).

$$similarity_{cosine}(w_1, w_2) = \frac{w_1 . w_2}{\|w_1\|_2 \|w_2\|_2}$$

In [2]:
```
### Let's build term-term co-occurrence matrix for our text.
sys.path.append("../code/.")

from comat import CooccurrenceMatrix
from preprocessing import MyPreprocessor

corpus = [
    "How tall is Machu Picchu?",
    "Machu Picchu is 13.164 degrees south of the equator.",
    "The official height of Machu Picchu is 2,430 m.",
    "Machu Picchu is 80 kilometres (50 miles) northwest of Cusco.",
    "It is 80 kilometres (50 miles) northwest of Cusco, on the crest of
]
pp = MyPreprocessor()
pp_corpus = pp.preprocess_corpus(corpus)
cm = CooccurrenceMatrix(pp_corpus)
vocab, comat = cm.fit_transform()
words = [
    key for key, value in sorted(vocab.items(), key=lambda item: (item[
]
df = pd.DataFrame(comat.todense(), columns=words, index=words, dtype=np
df.head()
```

Out[2]:

	tall	machu	picchu	13.164	degrees	south	equator	official	height	2,430	...	mean	se
tall	0	1	1	0	0	0	0	0	0	0	...	0	
machu	1	0	5	1	1	0	0	1	1	2	...	0	
picchu	1	5	0	1	1	1	0	1	1	2	...	0	
13.164	0	1	1	0	1	1	1	0	0	0	...	0	
degrees	0	1	1	1	0	1	1	0	0	0	...	0	

5 rows × 32 columns

In [3]:
```
from sklearn.metrics.pairwise import cosine_similarity
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

def similarity(word1, word2):
    """
    Returns similarity score between word1 and word2
    Arguments
    ---------
    word1 -- (str)
        The first word
    word2 -- (str)
        The second word

    Returns
    --------
    None. Prints the similarity score between word1 and word2.
    """
    vec1 = cm.get_word_vector(word1).todense().flatten()
    vec2 = cm.get_word_vector(word2).todense().flatten()
    v1 = np.squeeze(np.asarray(vec1))
    v2 = np.squeeze(np.asarray(vec2))
    print(
        "The dot product between %s and %s is %0.2f and cosine similari
        % (word1, word2, np.dot(v1, v2), cosine_similarity(vec1, vec2))
    )


similarity("tall", "height")
similarity("tall", "official")

### Not very reliable similarity scores because we used only 4 sentence
```

```
The dot product between tall and height is 2.00 and cosine similarity is
0.71
The dot product between tall and official is 2.00 and cosine similarity i
s 0.82
```

- We are able to capture some similarities between words now.
- That said similarities do not make much sense in the toy example above because we're using a tiny corpus.
- To find meaningful patterns of similarities between words, we need a large corpus.
- Let's try a bit larger corpus and check whether the similarities make sense.

In [4]:
```
import wikipedia
from nltk.tokenize import sent_tokenize, word_tokenize

corpus = []

queries = ["Machu Picchu", "human stature"]

for i in range(len(queries)):
    sents = sent_tokenize(wikipedia.page(queries[i]).content)
    corpus.extend(sents)
print("Number of sentences in the corpus: ", len(sents))
```

```
Number of sentences in the corpus:  269
```

In [5]:
```
pp = MyPreprocessor()
pp_corpus = pp.preprocess_corpus(corpus)
cm = CooccurrenceMatrix(pp_corpus)
vocab, comat = cm.fit_transform()
words = [
    key for key, value in sorted(vocab.items(), key=lambda item: (item[
]
df = pd.DataFrame(comat.todense(), columns=words, index=words, dtype=np
df.head()
```

Out[5]:

	machu	picchu	15th-century	inca	citadel	located	eastern	cordillera	southern	peru	...	cha
machu	0	81	1	5	1	2	1	0	1	2	...	
picchu	81	0	1	6	3	1	1	0	1	1	...	
15th-century	1	1	0	1	1	1	0	0	0	0	...	
inca	5	6	1	0	1	1	1	0	0	1	...	
citadel	1	3	1	1	0	1	1	1	0	0	...	

5 rows × 3019 columns

In [6]:
```
similarity("tall", "height")
similarity("tall", "official")
```

```
The dot product between tall and height is 72.00 and cosine similarity is
0.23
The dot product between tall and official is 0.00 and cosine similarity i
s 0.00
```

Sparse vs. dense word vectors

- Term-term co-occurrence matrices are long and sparse.
  - length |V| is usually large (e.g., > 50,000)

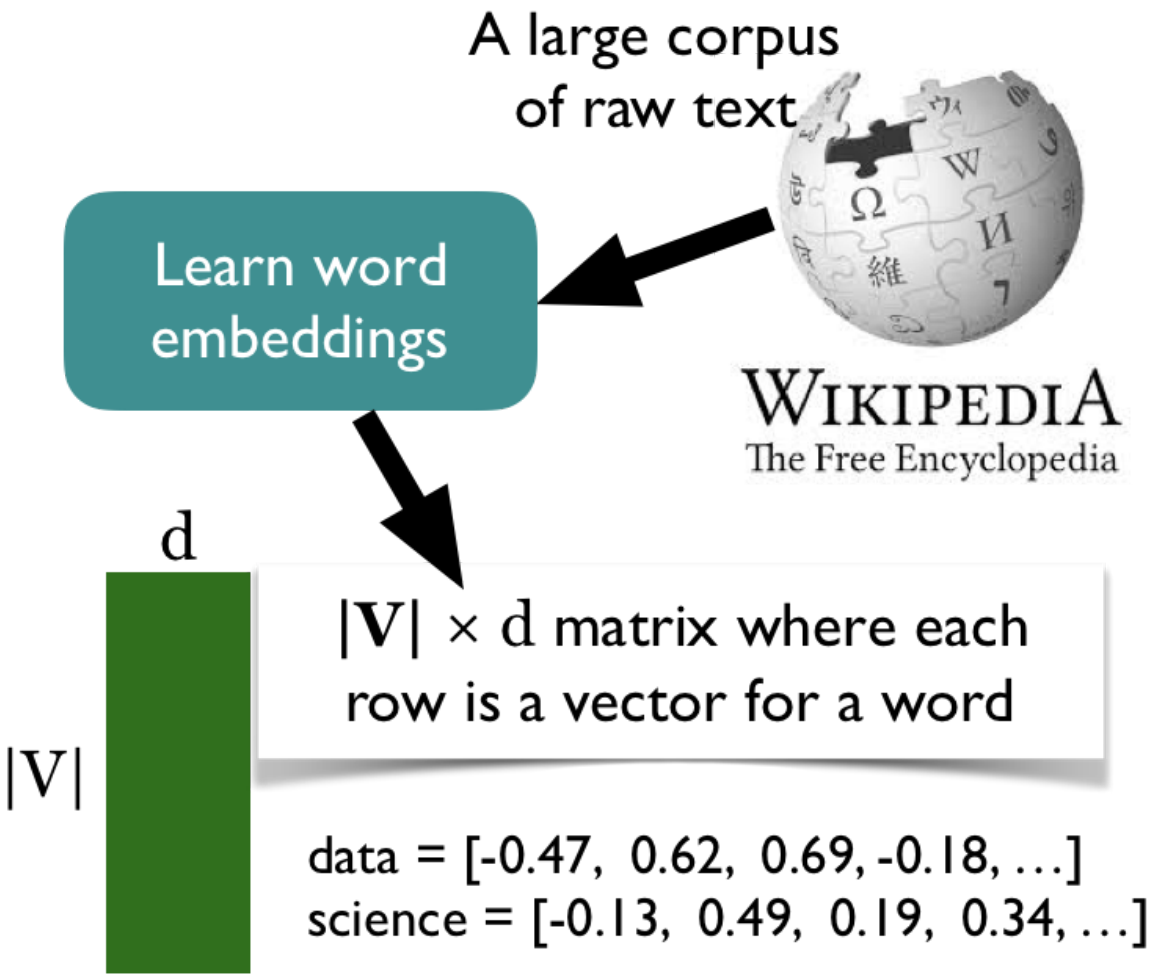
  - most elements are zero
- OK because there are efficient ways to deal with sparse matrices.

Alternative

- Learn short (~100 to 1000 dimensions) and dense vectors.
- Short vectors may be easier to train with ML models (less weights to train).
- They may generalize better.
- In practice they work much better!

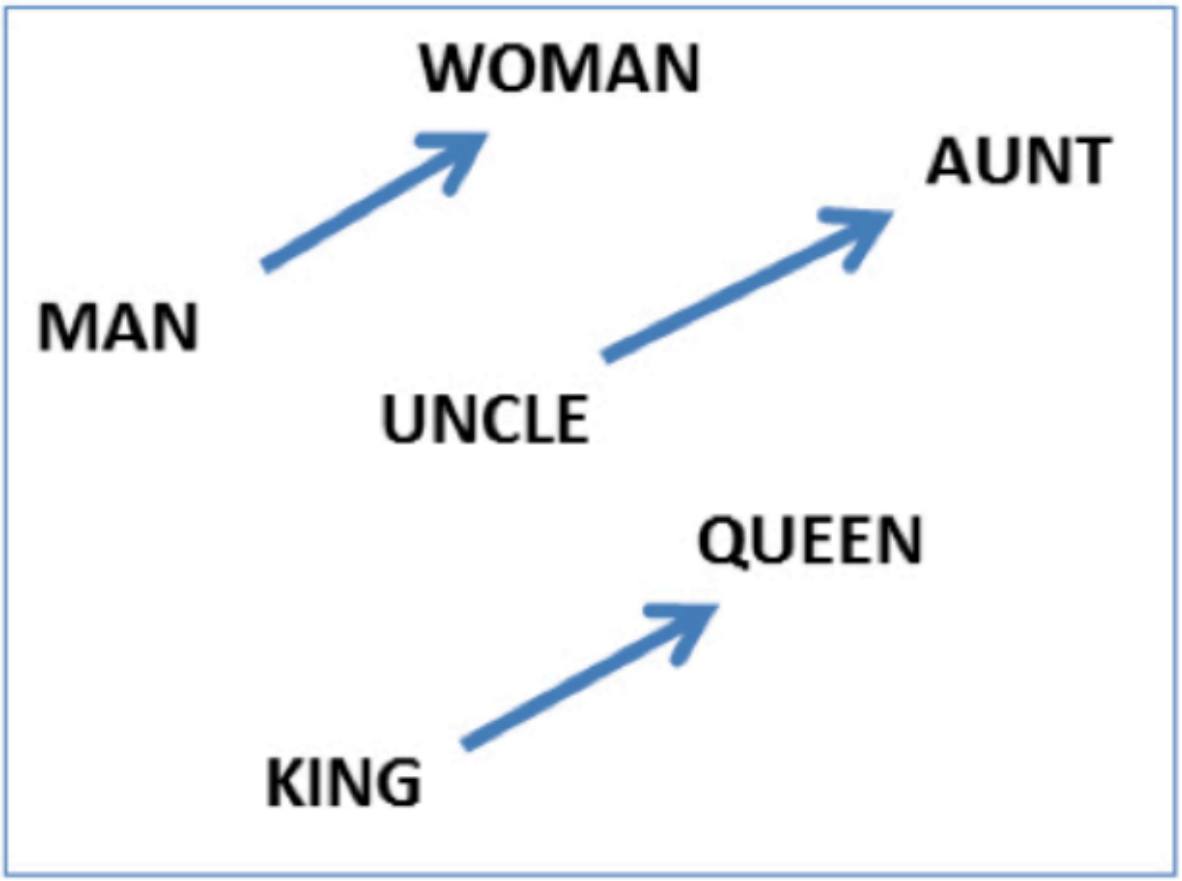
Word2Vec

- A family of algorithms to create dense word embeddings



Success of Word2Vec

- Able to capture complex relationships between words.
- Example: What is the word that is similar to **WOMAN** in the same sense as **KING** is similar to **MAN**?
- Perform a simple algebraic operations with the vector representation of words.
  $\vec{X} = \vec{\text{KING}} - \vec{\text{MAN}} + \vec{\text{WOMAN}}$
- Search in the vector space for the word closest to $\vec{X}$ measured by cosine distance.



(Credit: Mikolov et al. 2013)

- We can create a dense representation with a library called `gensim`.

  `conda install -c anaconda gensim`

In [7]:
```
from gensim.models import Word2Vec

sentences = [["cat", "say", "meow"], ["dog", "say", "woof"]]
model = Word2Vec(sentences, min_count=1)
```

Let's look at the word vector of the word *cat*.

In [8]:
```
model.wv["cat"]
```

Out[8]:
```
array([-0.00713902,  0.00124103, -0.00717672, -0.00224462,  0.0037193 ,
        0.00583312,  0.00119818,  0.00210273, -0.00411039,  0.00722533,
       -0.00630704,  0.00464722, -0.00821997,  0.00203647, -0.00497705,
       -0.00424769, -0.00310898,  0.00565521,  0.0057984 , -0.00497465,
        0.00077333, -0.00849578,  0.00780981,  0.00925729, -0.00274233,
        0.00080022,  0.00074665,  0.00547788, -0.00860608,  0.00058446,
        0.00686942,  0.00223159,  0.00112468, -0.00932216,  0.00848237,
       -0.00626413, -0.00299237,  0.00349379, -0.00077263,  0.00141129,
        0.00178199, -0.0068289 , -0.00972481,  0.00904058,  0.00619805,
       -0.00691293,  0.00340348,  0.00020606,  0.00475375, -0.00711994,
        0.00402695,  0.00434743,  0.00995737, -0.00447374, -0.00138926,
       -0.00731732, -0.00969783, -0.00908026, -0.00102275, -0.00650329,
        0.00484973, -0.00616403,  0.00251919,  0.00073944, -0.00339215,
       -0.00097922,  0.00997913,  0.00914589, -0.00446183,  0.00908303,
       -0.00564176,  0.00593092, -0.00309722,  0.00343175,  0.00301723,
        0.00690046, -0.00237388,  0.00877504,  0.00758943, -0.00954765,
       -0.00800821, -0.0076379 ,  0.00292326, -0.00279472, -0.00692952,
       -0.00812826,  0.00830918,  0.00199049, -0.00932802, -0.00479272,
        0.00313674, -0.00471321,  0.00528084, -0.00423344,  0.0026418 ,
       -0.00804569,  0.00620989,  0.00481889,  0.00078719,  0.00301345],
      dtype=float32)
```

What's the most similar word to the word *cat*?

In [9]:
```
model.wv.most_similar("cat")
```

Out[9]:
```
[('dog', 0.17018887400627136),
 ('woof', 0.004502999130636454),
 ('say', -0.027750356122851372),
 ('meow', -0.04461711645126343)]
```

This is good. But if you want good and meaningful representations of words you need to train models on a large corpus such as the whole Wikipedia, which is computationally intensive.

So instead of training our own models, we use the **pre-trained embeddings**. These are the word embeddings people have trained embeddings on huge corpora and made them available for us to use.

Let's try out Google news pre-trained word vectors.

In [10]:
```
# It'll take a while to run this when you try it out for the first time
import gensim.downloader as api

google_news_vectors = api.load("word2vec-google-news-300")
```

In [11]:
```
print("Size of vocabulary: ", len(google_news_vectors))
```

```
Size of vocabulary:  3000000
```

- `google_news_vectors` above has 300 dimensional word vectors for 3,000,000 unique words from Google news.

- What can we do with these word vectors?

Finding similar words

- Given word w, search in the vector space for the word closest to w as measured by cosine distance.

In [12]:
```
1 google_news_vectors.most_similar("UBC")
```

Out[12]:
```
[('UVic', 0.788647472858429),
 ('SFU', 0.7588527202606201),
 ('Simon_Fraser', 0.7356573939323425),
 ('UFV', 0.6880434155464172),
 ('VIU', 0.6778583526611328),
 ('Kwantlen', 0.6771427989006042),
 ('UBCO', 0.6734487414360046),
 ('UPEI', 0.6731125116348267),
 ('UBC_Okanagan', 0.6709135174751282),
 ('Lakehead_University', 0.662250816822052)]
```

In [13]:
```
1 google_news_vectors.most_similar("information")
```

Out[13]:
```
[('info', 0.7363681793212891),
 ('infomation', 0.680029571056366),
 ('infor_mation', 0.6733849048614502),
 ('informaiton', 0.6639008522033691),
 ('informa_tion', 0.6601257920265198),
 ('informationon', 0.633933424949646),
 ('informationabout', 0.6320980787277222),
 ('Information', 0.6186580657958984),
 ('informaion', 0.6093292236328125),
 ('details', 0.6063088178634644)]
```

If you want to extract all documents containing information or similar words, you could use this information.

Finding similarity scores between words

In [14]:
```
1 google_news_vectors.similarity("Canada", "hockey")
```

Out[14]:
```
0.27610135
```

In [15]:
```
1 google_news_vectors.similarity("China", "hockey")
```

Out[15]:
```
0.060384665
```

In [16]:
```
word_pairs = [
    ("height", "tall"),
    ("pineapple", "mango"),
    ("pineapple", "juice"),
    ("sun", "robot"),
    ("GPU", "lion"),
]
for pair in word_pairs:
    print(
        "The similarity between %s and %s is %0.3f"
        % (pair[0], pair[1], google_news_vectors.similarity(pair[0], pa
    )
```

```
The similarity between height and tall is 0.473
The similarity between pineapple and mango is 0.668
The similarity between pineapple and juice is 0.418
The similarity between sun and robot is 0.029
The similarity between GPU and lion is 0.002
```

In [17]:
```
def analogy(word1, word2, word3, model=google_news_vectors):
    """
    Returns analogy word using the given model.

    Parameters
    --------------
    word1 : (str)
        word1 in the analogy relation
    word2 : (str)
        word2 in the analogy relation
    word3 : (str)
        word3 in the analogy relation
    model :
        word embedding model

    Returns
    ---------------
        pd.dataframe
    """
    print("%s : %s :: %s : ?" % (word1, word2, word3))
    sim_words = model.most_similar(positive=[word3, word2], negative=[w
    return pd.DataFrame(sim_words, columns=["Analogy word", "Score"])
```

In [18]:
```
1 analogy("man", "king", "woman")
```

```
man : king :: woman : ?
```

Out[18]:

	Analogy word	Score
0	queen	0.711819
1	monarch	0.618967
2	princess	0.590243
3	crown_prince	0.549946
4	prince	0.537732
5	kings	0.523684
6	Queen_Consort	0.523595
7	queens	0.518113
8	sultan	0.509859
9	monarchy	0.508741

In [19]:
```
1 analogy("Montreal", "Canadiens", "Vancouver")
```

```
Montreal : Canadiens :: Vancouver : ?
```

Out[19]:

	Analogy word	Score
0	Canucks	0.821327
1	Vancouver_Canucks	0.750401
2	Calgary_Flames	0.705471
3	Leafs	0.695783
4	Maple_Leafs	0.691617
5	Thrashers	0.687503
6	Avs	0.681716
7	Sabres	0.665307
8	Blackhawks	0.664625
9	Habs	0.661023

In [20]:
```
1 analogy("Toronto", "UofT", "Vancouver")
```

```
Toronto : UofT :: Vancouver : ?
```

Out[20]:

	Analogy word	Score
0	SFU	0.579245
1	UVic	0.576921
2	UBC	0.571431
3	Simon_Fraser	0.543464
4	Langara_College	0.541347
5	UVIC	0.520495
6	Grant_MacEwan	0.517273
7	UFV	0.514150
8	Ubyssey	0.510421
9	Kwantlen	0.503807

Examples of semantic and syntactic relationships

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

(Credit: Mikolov 2013)

Implicit biases and stereotypes in word embeddings

- Reflect gender stereotypes present in broader society.

- They may also amplify these stereotypes because of their widespread usage.
- See the paper [Man is to Computer Programmer as Woman is to ... (http://papers.nips.cc/paper/6228-man-is-to-computer-programmer-as-woman-is-to-homemaker-debiasing-word-embeddings.pdf)](http://papers.nips.cc/paper/6228-man-is-to-computer-programmer-as-woman-is-to-homemaker-debiasing-word-embeddings.pdf).

In [21]:
```
1 analogy("man", "computer_programmer", "woman")
```

```
man : computer_programmer :: woman : ?
```

Out[21]:

	Analogy word	Score
0	homemaker	0.562712
1	housewife	0.510505
2	graphic_designer	0.505180
3	schoolteacher	0.497949
4	businesswoman	0.493489
5	paralegal	0.492551
6	registered_nurse	0.490797
7	saleswoman	0.488163
8	electrical_engineer	0.479773
9	mechanical_engineer	0.475540

Most of the modern embeddings are de-biased.

(NLP)### Other pre-trained embeddings

A number of pre-trained word embeddings are available. The most popular ones are:

- Word2Vec (https://code.google.com/archive/p/word2vec/)
  - trained on several corpora using the word2vec algorithm
- wikipedia2vec (https://wikipedia2vec.github.io/wikipedia2vec/pretrained/)
  - pretrained embeddings for 12 languages
- GloVe (https://nlp.stanford.edu/projects/glove/)
  - trained using the GloVe algorithm (https://nlp.stanford.edu/pubs/glove.pdf)
  - published by Stanford University
- fastText pre-trained embeddings for 294 languages (https://fasttext.cc/docs/en/pretrained-vectors.html)
  - trained using the fastText algorithm (http://aclweb.org/anthology/Q17-1010)
  - published by Facebook

```
Note that these pre-trained word vectors are of big size (in severa
l gigabytes).
```

Here is a list of all pre-trained embeddings available with `gensim`.

In [22]:
```
import gensim.downloader

print(list(gensim.downloader.info()["models"].keys()))
```

```
['fasttext-wiki-news-subwords-300', 'conceptnet-numberbatch-17-06-300',
'word2vec-ruscorpora-300', 'word2vec-google-news-300', 'glove-wiki-gigawo
rd-50', 'glove-wiki-gigaword-100', 'glove-wiki-gigaword-200', 'glove-wiki
-gigaword-300', 'glove-twitter-25', 'glove-twitter-50', 'glove-twitter-10
0', 'glove-twitter-200', '__testing_word2vec-matrix-synopsis']
```

Word vectors with spaCy

- spaCy also gives you access to word vectors with bigger models: `en_core_web_md` or `en_core_web_lr`
- `spaCy` 's pre-trained embeddings are trained on OntoNotes corpus (https://catalog.ldc.upenn.edu/LDC2013T19).
- This corpus has a collection of different styles of texts such as telephone conversations, newswire, newsgroups, broadcast news, broadcast conversation, weblogs, religious texts.
- Let's try it out.

In [23]:

```
import spacy

nlp = spacy.load("en_core_web_md")

doc = nlp("pineapple")
doc.vector
```

Out[23]:
```
array([ 6.5486e-01, -2.2584e+00,  6.2793e-02,  1.8801e+00,  2.0700e-01,
       -3.3299e+00, -9.6833e-01,  1.5131e+00, -3.7041e+00, -7.7749e-02,
        1.5029e+00, -1.7764e+00,  1.7324e+00,  1.6241e+00,  2.6455e-01,
       -3.0840e+00,  7.5715e-01, -1.2903e+00,  2.3571e+00, -3.8793e+00,
        7.7635e-01,  3.9372e+00,  3.9900e-01, -6.8284e-01, -1.4018e+00,
       -2.1673e+00, -1.9244e+00,  1.0629e+00,  3.3378e-01, -8.3864e-01,
       -2.5646e-01, -1.7198e+00, -5.4607e-02, -1.4614e+00,  1.3352e+00,
       -1.8177e+00,  1.7254e+00,  4.9624e-01,  1.1314e+00, -1.5295e+00,
       -8.8629e-01, -2.7562e-01,  7.1799e-01,  1.5554e-01,  3.4230e+00,
        2.7167e+00,  1.1793e+00,  2.0961e-01,  3.3121e-01,  1.2322e+00,
        1.4375e+00, -4.2099e-01,  6.2814e-01, -1.9051e+00,  3.0593e-02,
        6.1895e-01, -3.1495e-01, -2.0444e-04,  2.2073e+00,  3.8856e-01,
        1.6554e+00,  1.1932e+00,  2.6678e+00, -5.5454e-01, -1.2078e+00,
        1.5709e-01, -1.1324e+00, -2.0163e+00,  1.4567e+00, -2.4244e-01,
       -1.9425e+00,  8.3090e-01,  1.7428e-01,  9.1676e-01,  8.8830e-03,
        2.4857e-01, -1.2018e+00, -2.3073e+00,  2.2553e+00, -1.5853e+00,
       -5.8452e-01,  9.2523e-01, -2.7129e-01, -7.6348e-01,  1.3506e+00,
        1.7429e+00,  3.0469e+00,  1.9319e+00, -2.6099e+00,  1.8484e+00,
        1.3795e+00,  2.0948e+00,  1.1545e+00, -2.9681e+00, -5.0455e-02,
       -5.3864e-01,  2.4820e+00, -1.1131e+00, -2.1827e-01, -2.7559e+00,
       -4.4502e-01, -2.8897e+00,  1.7430e+00, -1.5742e+00,  5.7160e-02,
        2.4764e+00, -2.5828e+00,  9.3866e-01, -1.3150e+00,  2.3863e+00,
        6.1536e-01,  1.7656e-01,  2.0245e+00,  1.6807e-01, -1.2850e+00,
        1.6425e-01,  1.7782e+00, -3.2221e+00,  6.1392e-01,  1.3269e+00,
       -3.1582e-02,  6.6331e-01, -6.8109e-01,  5.0985e-01, -4.2942e-01,
       -1.6438e-01,  7.9306e-01, -3.0776e+00,  1.8022e+00, -4.5356e-01,
       -1.6405e+00,  8.1761e-01,  1.4960e+00, -6.2266e-01,  8.5264e-01,
       -5.0226e-01, -1.5735e+00, -4.5090e+00, -5.0587e-01, -1.5471e+00,
       -5.3910e-01, -6.6574e-01,  7.6376e-01, -1.4926e+00, -7.8819e-01,
       -9.9256e-01,  1.1512e+00,  5.2091e-01,  1.6460e-01, -2.6747e+00,
       -1.7082e+00,  1.5789e+00, -2.8982e-01, -1.2842e+00, -1.1286e+00,
        7.6392e-01,  3.2199e+00,  7.5850e-01,  1.3628e+00, -1.3231e+00,
        2.2350e-02, -2.5602e+00,  6.7751e-01,  4.0511e-01,  1.8997e+00,
       -1.1051e+00, -1.3014e-01,  7.2024e-01,  6.2354e-02,  1.1913e-01,
       -1.1978e+00, -1.5625e+00, -2.5975e-01,  2.5911e+00, -3.2413e+00,
       -3.8988e-01, -4.0542e-01, -1.8894e+00,  3.4278e+00, -3.3625e-01,
       -2.0979e+00,  1.3275e+00, -2.0514e+00,  2.4583e-01, -7.3326e-01,
       -2.3684e+00,  2.8493e+00, -5.2075e-01,  2.2708e-01, -6.8701e-01,
       -7.0855e-01, -7.5334e-01,  7.3050e-02,  2.2246e+00, -2.6824e-01,
       -2.8289e-01, -1.8230e+00,  2.2047e+00, -2.4848e-01, -2.3042e-02,
        1.0358e+00, -2.7074e-01, -5.6816e-02, -9.1017e-01,  1.2943e-01,
       -1.4274e+00, -3.6128e-01,  7.3127e-01,  2.0264e+00,  7.2928e-01,
        1.7298e+00,  1.1075e+00, -7.0250e-01,  1.6928e+00,  2.0074e+00,
       -7.5464e-01,  1.6378e+00,  3.5970e-01, -2.2128e-01, -1.7607e-01,
        1.8260e+00, -2.5962e-01, -1.4320e+00,  7.8332e-01,  2.1438e+00,
       -2.4723e+00, -1.4913e-01,  6.2585e-01,  6.6819e-01,  2.3947e+00,
       -2.7173e+00,  2.4134e-03, -8.6530e-01, -9.7728e-01, -2.9815e+00,
        1.6895e+00, -7.1146e-01,  3.2025e+00, -9.4129e-01, -1.9695e+00,
        7.7711e-01, -3.2278e-01, -1.3727e+00,  2.9276e+00, -1.5440e-01,
        1.7169e+00,  5.5736e-01,  1.4620e-01, -1.1244e+00, -2.4633e+00,
       -2.2685e+00,  1.2459e+00, -2.0362e+00, -4.8331e-01, -6.3194e-01,
       -2.4082e+00, -9.0132e-01,  3.0541e+00, -2.2632e+00, -3.7800e-01,
       -3.1647e-01,  1.0785e+00, -3.0444e-01,  1.2112e+00, -1.3496e+00,
        1.0599e+00,  4.2607e-01,  4.0194e-01, -2.8586e+00,  1.0107e+00,
        1.5924e+00, -5.1770e-01,  1.3246e+00,  3.2268e-01, -1.3978e-01,
       -2.1841e+00,  1.6548e+00,  1.3903e+00,  6.3376e-01, -4.7083e-01,
        6.8377e-01, -1.3031e+00, -1.3292e-01, -1.1567e+00,  5.3419e-01,
```

```
       -1.3412e+00, -1.5887e+00, -9.4468e-01, -2.4031e+00,  3.1785e+00,
        1.1524e+00, -1.1699e+00,  9.8752e-01, -1.0660e+00, -2.1852e+00,
       -3.1228e-01,  3.0012e+00, -1.2234e+00,  5.7454e-01, -2.1885e-01],
      dtype=float32)
```

Representing documents using word embeddings

- Assuming that we have reasonable representations of words.
- How do we represent meaning of paragraphs or documents?
- Two simple approaches
  - Averaging embeddings
  - Concatenating embeddings

Averaging embeddings

> All empty promises

$(embedding(all) + embedding(empty) + embedding(promise))/3$

Average embeddings with spaCy

- We can do this conveniently with spaCy (https://spacy.io/usage/linguistic-features#vectors-similarity).
- We need `en_core_web_md` model to access word vectors.
- You can download the model by going to command line and in your course `conda` environment and download `en_core_web_md` as follows.

```
conda activate cpsc330
python -m spacy download en_core_web_md
```

We can access word vectors for individual words in `spaCy` as follows.

In [24]:
```
nlp("empty").vector[0:10]
```

Out[24]:
```
array([ 0.010289,  4.9203  , -0.48081 ,  3.5738  , -2.2516  ,  2.1697  ,
       -1.0116  ,  2.4216  , -3.7343  ,  3.3025  ], dtype=float32)
```

We can get average embeddings for a sentence or a document in `spaCy` as follows:

In [25]:
```
s = "All empty promises"
doc = nlp(s)
avg_sent_emb = doc.vector
print(avg_sent_emb.shape)
print("Vector for: {}\n{}".format((s), (avg_sent_emb[0:10])))
```

```
(300,)
Vector for: All empty promises
[-0.459937    1.9785299   1.0319      1.5123      1.4806334   2.73183
  1.204       1.1724668  -3.5227966  -0.05656664]
```

Similarity between documents

- We can also get similarity between documents as follows.
- Note that this is based on average embeddings of each sentence.

In [80]:
```
doc1 = nlp("Deep learning is very popular these days.")
doc2 = nlp("Machine learning is dominated by neural networks.")
doc3 = nlp("A home-made fresh bread with butter and cheese.")

# Similarity of two documents
print(doc1, "<->", doc2, doc1.similarity(doc2))
print(doc2, "<->", doc3, doc2.similarity(doc3))
```

```
Deep learning is very popular these days. <-> Machine learning is dominat
ed by neural networks. 0.6998688700400709
Machine learning is dominated by neural networks. <-> A home-made fresh b
read with butter and cheese. 0.5098292895236649
```

- Do these scores make sense?
- There are no common words, but we are still able to identify that doc1 and doc2 are more similar that doc2 and doc3.
- You can use such average embedding representation in text classification tasks.

Airline sentiment analysis using average embedding representation

- Let's try average embedding representation for airline sentiment analysis.
- You can download the dataset here (https://www.kaggle.com/jaskarancr/airline-sentiment-dataset).

In [27]:
```
df = pd.read_csv("../data/Airline-Sentiment-2-w-AA.csv", encoding="ISO-
```

In [28]:
```
from sklearn.model_selection import cross_validate, train_test_split

train_df, test_df = train_test_split(df, test_size=0.2, random_state=12
X_train, y_train = train_df["text"], train_df["airline_sentiment"]
X_test, y_test = test_df["text"], test_df["airline_sentiment"]
```

In [29]:
```
train_df.head()
```

Out[29]:

	_unit_id	_golden	_unit_state	_trusted_judgments	_last_judgment_at	airline_sentiment	airl
5789	681455792	False	finalized	3	2/25/15 4:21	negative	
8918	681459957	False	finalized	3	2/25/15 9:45	neutral	
11688	681462990	False	finalized	3	2/25/15 9:53	negative	
413	681448905	False	finalized	3	2/25/15 10:10	neutral	
4135	681454122	False	finalized	3	2/25/15 10:08	negative	

Bag-of-words representation for sentiment analysis

In [30]:
```
pipe = make_pipeline(
    CountVectorizer(stop_words="english"), LogisticRegression(max_iter=
)
pipe.named_steps["countvectorizer"].fit(X_train)
X_train_transformed = pipe.named_steps["countvectorizer"].transform(X_t
print("Data matrix shape:", X_train_transformed.shape)
pipe.fit(X_train, y_train);
```

```
Data matrix shape: (11712, 13064)
```

In [31]:
```
print("Train accuracy {:.2f}".format(pipe.score(X_train, y_train)))
print("Test accuracy {:.2f}".format(pipe.score(X_test, y_test)))
```

```
Train accuracy 0.94
Test accuracy 0.80
```

Sentiment analysis with average embedding representation

- Let's see how can we get word vectors using `spaCy` .
- Let's create average embedding representation for each example.

In [32]:
```
X_train_embeddings = pd.DataFrame([text.vector for text in nlp.pipe(X_t
X_test_embeddings = pd.DataFrame([text.vector for text in nlp.pipe(X_te
```

We have reduced dimensionality from 13,064 to 300!

In [33]:
```
X_train_embeddings.shape
```

Out[33]:
```
(11712, 300)
```

In [34]:
```
X_train_embeddings.head()
```

Out[34]:

	0	1	2	3	4	5	6	7	8	
0	1.259942	4.856640	-2.677500	-1.875390	0.459240	-0.304300	3.273020	2.008458	-4.818360	1
1	-0.563826	0.869816	-2.462877	0.057751	0.892089	0.566698	-0.283121	3.594112	-1.578107	1
2	-0.707503	0.908782	-3.248327	-0.667797	2.590958	2.246804	-1.095754	3.953429	-1.524224	0
3	-0.977736	4.676425	-0.224362	-1.011286	3.981441	-1.132660	0.988456	2.997113	0.553975	-0
4	-0.725984	0.178514	-1.163662	0.597000	4.621603	-1.166608	1.256955	3.940082	0.530063	-0

5 rows × 300 columns

Sentiment classification using average embeddings

- What are the train and test accuracies with average word embedding representation?
- The accuracy is a bit better with less overfitting.
- Note that we are using **transfer learning** here.
- The embeddings are trained on a completely different corpus.

In [35]:
```
lgr = LogisticRegression(max_iter=1000, C=.1)
lgr.fit(X_train_embeddings, y_train)
print("Train accuracy {:.2f}".format(lgr.score(X_train_embeddings, y_tr
print("Test accuracy {:.2f}".format(lgr.score(X_test_embeddings, y_test
```

```
Train accuracy 0.80
Test accuracy 0.79
```

Sentiment classification using advanced sentence representations

- Since representing documents is so essential for text classification tasks, there are more advanced methods for document representation.

We will use: `conda install -c conda-forge sentence-transformers`

In [36]:
```
from sentence_transformers import SentenceTransformer

embedder = SentenceTransformer("sentence-transformers/paraphrase-distil
```

In [37]:
```
emb_sents = embedder.encode("all empty promises")
emb_sents.shape
```

Out[37]: (768,)

In [38]:
```
emb_train = embedder.encode(train_df["text"].tolist())
emb_train_df = pd.DataFrame(emb_train, index=train_df.index)
emb_train_df
```

Out[38]:

	0	1	2	3	4	5	6	7	
5789	-0.120494	0.250263	-0.022795	-0.116368	0.078650	0.037357	-0.251341	0.321429	-0.143
8918	-0.182954	0.118282	0.066341	-0.136098	0.094947	-0.121303	0.069233	-0.097500	0.025
11688	-0.032988	0.630251	-0.079516	0.148981	0.194708	-0.226263	-0.043630	0.217398	-0.010
413	-0.119258	0.172168	0.098697	0.319858	0.415475	0.248359	-0.025923	0.385350	0.066
4135	0.094240	0.360193	0.213747	0.363690	0.275521	0.134936	-0.276319	0.009336	-0.021
...	...	...	...	...	...	...	...	...	
5218	-0.204408	-0.145289	-0.064201	0.213571	-0.140225	0.338556	-0.148578	0.224515	-0.042
12252	0.108408	0.438293	0.216812	-0.349289	0.422689	0.377760	0.045198	-0.034095	0.427
1346	0.068411	0.017591	0.236154	0.221446	-0.103568	0.055510	0.062910	0.067424	-0.003
11646	-0.091488	-0.155708	0.032391	0.018314	0.524997	0.563933	-0.080985	0.097982	-0.535
3582	0.185626	0.092904	0.097085	-0.174650	-0.193584	0.047294	0.098216	0.332670	0.163

11712 rows × 768 columns

In [39]:
```
emb_test = embedder.encode(test_df["text"].tolist())
emb_test_df = pd.DataFrame(emb_test, index=test_df.index)
emb_test_df
```

Out[39]:

	0	1	2	3	4	5	6	7	
1671	-0.002864	0.217326	0.124349	-0.082548	0.709688	-0.582441	0.257897	0.169356	0.2488
10951	-0.141048	0.137934	0.131319	0.194773	0.868204	0.078791	-0.131656	0.036244	-0.2157
5382	-0.252943	0.527507	-0.065608	0.013467	0.207989	0.003881	-0.066281	0.253166	0.0210
3954	0.054319	0.096738	0.113037	0.032039	0.493064	-0.641102	0.078760	0.402187	0.1897
11193	-0.065858	0.223270	0.507333	0.266193	0.104696	-0.219555	0.146247	0.315649	-0.1261
...	...	...	...	...	...	...	...	...	
5861	0.077512	0.322276	0.026697	-0.111393	0.174207	0.235201	0.053888	0.244942	0.1816
3627	-0.173311	-0.023604	0.190388	-0.136543	-0.360269	-0.444686	0.056311	0.291941	-0.3997
12559	-0.124635	-0.101799	0.129061	0.636907	0.681090	0.399300	-0.078321	0.221824	-0.2772
8123	0.063508	0.332506	0.119605	-0.001363	-0.161802	-0.082302	-0.025883	0.048027	0.1269
210	0.015537	0.425568	0.350672	0.113120	-0.128615	0.098112	0.222081	0.101654	0.2240

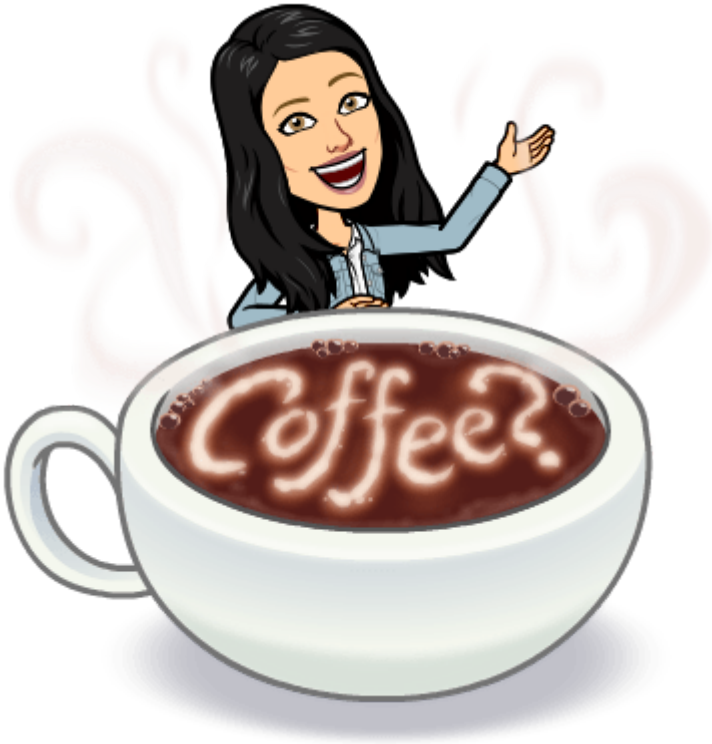
2928 rows × 768 columns

In [40]:
```
lgr = LogisticRegression(max_iter=1000)
lgr.fit(emb_train, y_train)
print("Train accuracy {:.2f}".format(lgr.score(emb_train, y_train)))
print("Test accuracy {:.2f}".format(lgr.score(emb_test, y_test)))
```

```
Train accuracy 0.87
Test accuracy 0.83
```

- Some improvement over bag of words and average embedding representations!
- But much slower ...

Break (5 min)

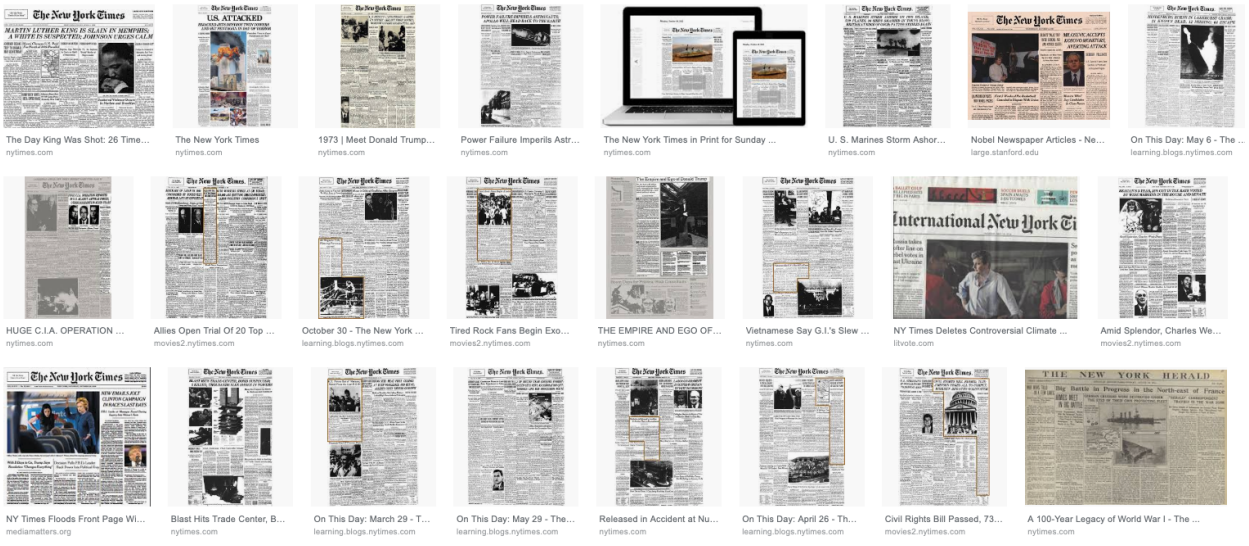


Topic modeling

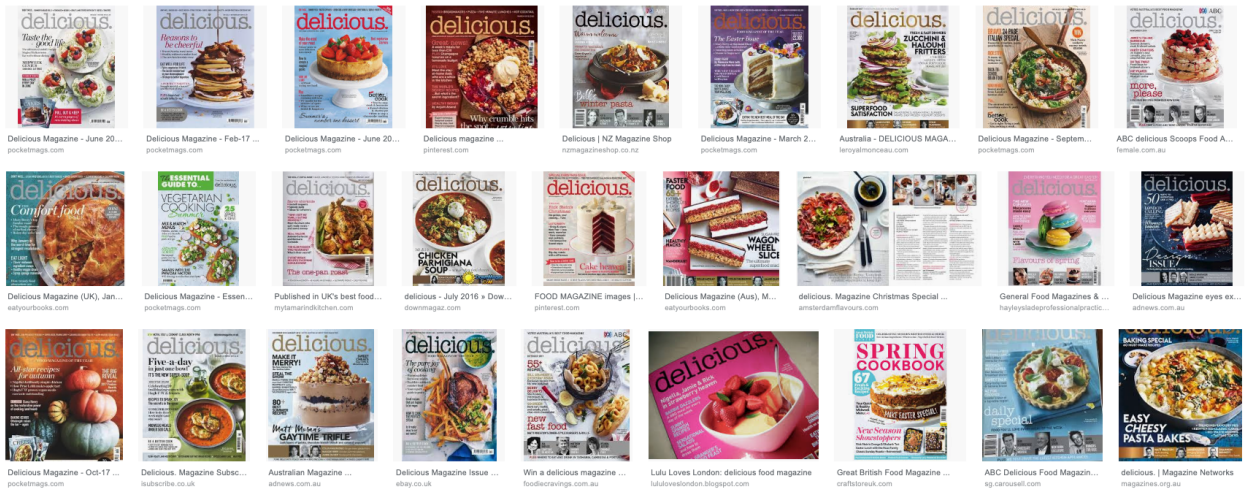
Topic modeling motivation

- Suppose you have a large collection of documents on a variety of topics.

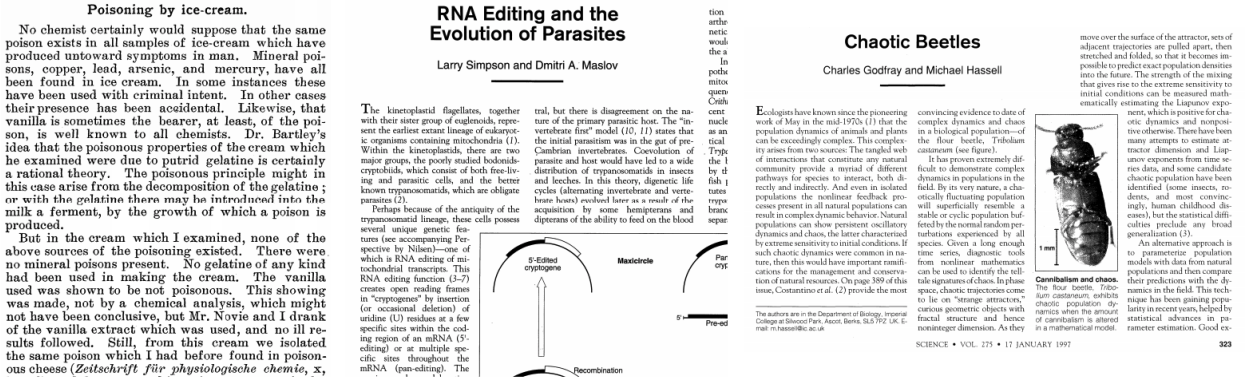
Example: A corpus of news articles



Example: A corpus of food magazines



A corpus of scientific articles



Poisoning by ice-cream.

No chemist certainly would suppose that the same poison exists in all samples of ice-cream which have produced untoward symptoms in man. Mineral poisons, copper, lead, arsenic, and mercury, have all been found in ice cream. In some instances these have been used with criminal intent. In other cases their presence has been accidental. Likewise, that vanilla is sometimes the bearer, at least, of the poison, is well known to all chemists. Dr. Bartley's idea that the poisonous properties of the cream which he examined were due to putrid gelatine is certainly a rational theory. The poisonous principle might in this case arise from the decomposition of the gelatine; or with the gelatine there may be introduced into the milk a ferment, by the growth of which a poison is produced.

But in the cream which I examined, none of the above sources of the poisoning existed. There were no mineral poisons present. No gelatine of any kind had been used in making the cream. The vanilla used was shown to be not poisonous. This showing was made, not by a chemical analysis, which might not have been conclusive, but Mr. Novie and I drank of the vanilla extract which was used, and no ill results followed. Still, from this cream we isolated the same poison which I had before found in poisonous cheese (*Zeitschrift für physiologische chemie*, x,

RNA Editing and the Evolution of Parasites

Larry Simpson and Dmitri A. Maslov

The kinetoplastid flagellates, together with their sister group of euglenoids, represent the earliest extant lineage of eukaryotic organisms containing mitochondria (1). Within the kinetoplastids, there are two major groups, the poorly studied bodonids-cryptobids, which consist of both free-living and parasitic cells, and the better known trypanosomatids, which are obligate parasites (2).

Perhaps because of the antiquity of the trypanosomatid lineage, these cells possess several unique genetic features (see accompanying Perspective by Nilsen)—one of which is RNA editing of mitochondrial transcripts. This RNA editing function (3–7) creates open reading frames in "cryptogenes" by insertion (or occasional deletion) of uridine (U) residues at a few specific sites within the coding region of an mRNA (5′-editing) or at multiple specific sites throughout the mRNA (pan-editing). The

tion
arthe
wou
the a
In
pothe
mito
quen
Crith
cent
nucle
as an
tical
Tryp
the l
by d
fish
tutes
trypa
bran
separ

5′-Edited cryptogene
Maxicircle
Pre-ed
Recombination

Chaotic Beetles

Charles Godfray and Michael Hassell

Ecologists have known since the pioneering work of May in the mid-1970s (1) that the population dynamics of animals and plants can be exceedingly complex. This complexity arises from two sources: The tangled web of interactions that constitute any natural community provide a myriad of different pathways for species to interact, both directly and indirectly. And even in isolated populations the nonlinear feedback processes present in all natural populations can result in complex dynamic behavior. Natural populations can show persistent oscillatory dynamics and chaos, the latter characterized by extreme sensitivity to initial conditions. If such chaotic dynamics were common in nature, then this would have important ramifications for the management and conservation of natural resources. On page 389 of this issue, Costantino *et al.* (2) provide the most convincing evidence to date of complex dynamics and chaos in a biological population—of the flour beetle, *Tribolium castaneum* (see figure).

It has proven extremely difficult to demonstrate complex dynamics in populations in the field. By its very nature, a chaotically fluctuating population will superficially resemble a stable or cyclic population buffeted by the normal random perturbations experienced by all species. Given a long enough time series, diagnostic tools from nonlinear mathematics can be used to identify the telltale signatures of chaos. In phase space, chaotic trajectories come to lie on "strange attractors," curious geometric objects with fractal structure and hence noninteger dimension. As they move over the surface of the attractor, sets of adjacent trajectories are pulled apart, then stretched and folded, so that it becomes impossible to predict exact population densities into the future. The strength of the mixing that gives rise to the extreme sensitivity to initial conditions can be measured mathematically estimating the Liapunov exponent, which is positive for chaotic dynamics and nonpositive otherwise. There have been many attempts to estimate attractor dimension and Liapunov exponents from time series data, and some candidate chaotic population have been identified (some insects, rodents, and most convincingly, human childhood diseases), but the statistical difficulties preclude any broad generalization (3).

An alternative approach is to parameterize population models with data from natural populations and then compare their predictions with the dynamics in the field. This technique has been gaining popularity in recent years, helped by statistical advances in parameter estimation. Good ex-

1 mm

Cannibalism and chaos. The flour beetle, *Tribolium castaneum*, exhibits chaotic population dynamics when the amount of cannibalism is altered in a mathematical model.

The authors are in the Department of Biology, Imperial College at Silwood Park, Ascot, Berks. SL5 7PZ, UK. E-mail: m.hassell@ic.ac.uk

(Credit: Dave Blei's presentation (http://www.cs.columbia.edu/~blei/talks/Blei_Science_2008.pdf))

Topic modeling motivation

- Humans are pretty good at reading and understanding a document and answering questions such as
  - What is it about?
  - Which documents is it related to?
- But for a large collection of documents it would take years to read all documents and organize and categorize them so that they are easy to search.
- You need an automated way
  - to get an idea of what's going on in the data or
  - to pull documents related to a certain topic

Topic modeling

- Topic modeling gives you an ability to summarize the major themes in a large collection of documents (corpus).
  - Example: The major themes in a collection of news articles could be
    - **politics**
    - **entertainment**
    - **sports**
    - **technology**
    - ...
- A common tool to solve such problems is unsupervised ML methods.
- Given the hyperparameter K, the idea of topic modeling is to describe the data using K "topics"

Topic modeling: Input and output

- Input
  - A large collection of documents
  - A value for the hyperparameter K (e.g., $K = 3$)
- Output
  1. Topic-words association
     - For each topic, what words describe that topic?
  2. Document-topics association
     - For each document, what topics are expressed by the document?

Topic modeling: Example

- Topic-words association
  - For each topic, what words describe that topic?
  - A topic is a mixture of words.

TOPIC 1	TOPIC 2	TOPIC 3
probabilistic, bayesian, markov, stationary, model, hidden, word2vec,	fashion, clothes, model, elegant, style, pattern, fresh, designer,	nutrition, health, butter, soup, bread, baking, apple, fresh,

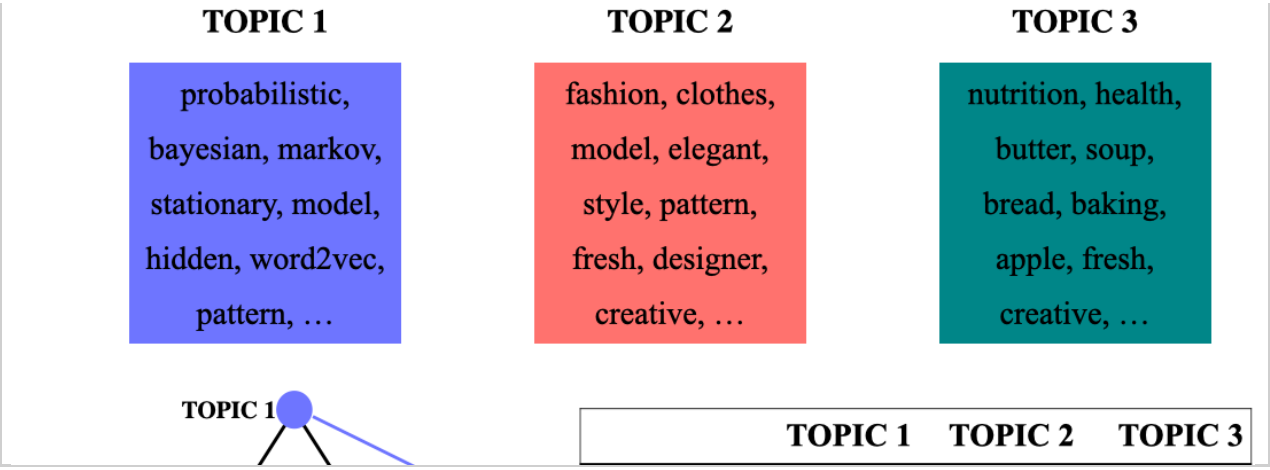
Topic modeling: Example

- Document-topics association
  - For each document, what topics are expressed by the document?
  - A document is a mixture of topics.

	TOPIC 1	TOPIC 2	TOPIC 3
Document 1	10%	30%	60%
Document 2	100%	0%	0%
Document 3	0%	100%	0%
Document 4	30%	30%	40%
...	...	...	...

Topic modeling: Input and output

- Input
  - A large collection of documents
  - A value for the hyperparameter K (e.g., $K = 3$)
- Output
  - For each topic, what words describe that topic?
  - For each document, what topics are expressed by the document?



Topic modeling: Some applications

- Topic modeling is a great EDA tool to get a sense of what's going on in a large corpus.
- Some examples
  - If you want to pull documents related to a particular lawsuit.
  - You want to examine people's sentiment towards a particular candidate and/or political party and so you want to pull tweets or Facebook posts related to election.

Topic modeling toy example

In [41]:
```
toy_df = pd.read_csv("../data/toy_lda_data.csv")
toy_df
```

Out[41]:

	doc_id	text
0	1	famous fashion model
1	2	fashion model pattern
2	3	fashion model probabilistic topic model confer...
3	4	famous fashion model
4	5	fresh fashion model
5	6	famous fashion model
6	7	famous fashion model
7	8	famous fashion model
8	9	famous fashion model
9	10	creative fashion model
10	11	famous fashion model
11	12	famous fashion model
12	13	fashion model probabilistic topic model confer...
13	14	probabilistic topic model
14	15	probabilistic model pattern
15	16	probabilistic topic model
16	17	probabilistic topic model
17	18	probabilistic topic model
18	19	probabilistic topic model
19	20	probabilistic topic model
20	21	probabilistic topic model
21	22	fashion model probabilistic topic model confer...
22	23	apple kiwi nutrition
23	24	kiwi health nutrition
24	25	fresh apple health
25	26	probabilistic topic model
26	27	creative health nutrition
27	28	probabilistic topic model
28	29	probabilistic topic model
29	30	hidden markov model probabilistic
30	31	probabilistic topic model
31	32	probabilistic topic model
32	33	apple kiwi nutrition
33	34	apple kiwi health
34	35	apple kiwi nutrition

	doc_id	text
35	36	fresh kiwi health
36	37	apple kiwi nutrition
37	38	apple kiwi nutrition
38	39	apple kiwi nutrition

In [42]:
```
from gensim import corpora, matutils, models

corpus = [doc.split() for doc in toy_df["text"].tolist()]
# Create a vocabulary for the lda (Latent Dirichlet Allocation) model
dictionary = corpora.Dictionary(corpus)
# Convert our corpus into document-term matrix for Lda
doc_term_matrix = [dictionary.doc2bow(doc) for doc in corpus]
```

In [43]:
```
from gensim.models import LdaModel

# Train an lda model
lda = models.LdaModel(
    corpus=doc_term_matrix,
    id2word=dictionary,
    num_topics=3,
    random_state=123,
    passes=10,
)
```

In [44]:
```
### Examine the topics in our LDA model
lda.print_topics(num_words=4)
```

Out[44]:
```
[(0,
  '0.303*"model" + 0.296*"probabilistic" + 0.261*"topic" + 0.040*"patter
n"'),
 (1, '0.245*"kiwi" + 0.219*"apple" + 0.218*"nutrition" + 0.140*"healt
h"'),
 (2, '0.308*"fashion" + 0.307*"model" + 0.180*"famous" + 0.071*"conferenc
e"')]
```

In [45]:
```
### Examine the topic distribution for a document
print("Document: ", corpus[0])
df = pd.DataFrame(lda[doc_term_matrix[0]], columns=["topic id", "probab
df.sort_values("probability", ascending=False)
```

```
Document:  ['famous', 'fashion', 'model']
```

Out[45]:

	topic id	probability
2	2	0.828760
0	0	0.087849
1	1	0.083391

You can also visualize the topics using `pyLDAvis`.

```
pip install pyLDAvis
```

> Do not install it using `conda`. They have made some changes in the recent version and `conda` build is not available for this version yet.

In [46]:

```
# Visualize the topics
import pyLDAvis

pyLDAvis.enable_notebook()
import pyLDAvis.gensim_models as gensimvis

vis = gensimvis.prepare(lda, doc_term_matrix, dictionary, sort_topics=F
vis
```

```
huggingface/tokenizers: The current process just got forked, after parall
elism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
	- Avoid using `tokenizers` before the fork if possible
	- Explicitly set the environment variable TOKENIZERS_PARALLELISM=
(true | false)
huggingface/tokenizers: The current process just got forked, after parall
elism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
	- Avoid using `tokenizers` before the fork if possible
	- Explicitly set the environment variable TOKENIZERS_PARALLELISM=
(true | false)
huggingface/tokenizers: The current process just got forked, after parall
elism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
	- Avoid using `tokenizers` before the fork if possible
	- Explicitly set the environment variable TOKENIZERS_PARALLELISM=
(true | false)
huggingface/tokenizers: The current process just got forked, after parall
elism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
	- Avoid using `tokenizers` before the fork if possible
	- Explicitly set the environment variable TOKENIZERS_PARALLELISM=
(true | false)
huggingface/tokenizers: The current process just got forked, after parall
elism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
	- Avoid using `tokenizers` before the fork if possible
	- Explicitly set the environment variable TOKENIZERS_PARALLELISM=
(true | false)
huggingface/tokenizers: The current process just got forked, after parall
elism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
	- Avoid using `tokenizers` before the fork if possible
	- Explicitly set the environment variable TOKENIZERS_PARALLELISM=
(true | false)
huggingface/tokenizers: The current process just got forked, after parall
elism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
	- Avoid using `tokenizers` before the fork if possible
	- Explicitly set the environment variable TOKENIZERS_PARALLELISM=
(true | false)
huggingface/tokenizers: The current process just got forked, after parall
elism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
	- Avoid using `tokenizers` before the fork if possible
	- Explicitly set the environment variable TOKENIZERS_PARALLELISM=
(true | false)
huggingface/tokenizers: The current process just got forked, after parall
elism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
	- Avoid using `tokenizers` before the fork if possible
	- Explicitly set the environment variable TOKENIZERS_PARALLELISM=
(true | false)
huggingface/tokenizers: The current process just got forked, after parall
elism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
```

```
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=
(true | false)
huggingface/tokenizers: The current process just got forked, after parall
elism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=
(true | false)
huggingface/tokenizers: The current process just got forked, after parall
elism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=
(true | false)
huggingface/tokenizers: The current process just got forked, after parall
elism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=
(true | false)
```

Out[46]:

Topic modeling pipeline

- Preprocess your corpus.
- Train LDA using `Gensim`.
- Interpret your topics.

Data

In [47]:
```
import wikipedia

queries = [
    "Artificial Intelligence",
    "unsupervised learning",
    "Supreme Court of Canada",
    "Peace, Order, and Good Government",
    "Canadian constitutional law",
    "ice hockey",
]
wiki_dict = {"wiki query": [], "text": []}
for i in range(len(queries)):
    wiki_dict["text"].append(wikipedia.page(queries[i]).content)
    wiki_dict["wiki query"].append(queries[i])

wiki_df = pd.DataFrame(wiki_dict)
wiki_df
```

Out[47]:

	wiki query	text
0	Artificial Intelligence	Artificial intelligence (AI) is intelligence—p...
1	unsupervised learning	Supervised learning (SL) is a machine learning...
2	Supreme Court of Canada	The Supreme Court of Canada (SCC; French: Cour...
3	Peace, Order, and Good Government	In many Commonwealth jurisdictions, the phrase...
4	Canadian constitutional law	Canadian constitutional law (French: droit con...
5	ice hockey	Ice hockey (or simply hockey) is a team sport ...

Preprocessing the corpus

- **Preprocessing is crucial!**
- Tokenization, converting text to lower case
- Removing punctuation and stopwords
- Discarding words with length < threshold or word frequency < threshold
- Possibly lemmatization: Consider the lemmas instead of inflected forms.
- Depending upon your application, restrict to specific part of speech;
    - For example, only consider nouns, verbs, and adjectives

We'll use `spaCy` (https://spacy.io/) for preprocessing.

In [48]:
```
import spacy

nlp = spacy.load("en_core_web_md", disable=["parser", "ner"])
```

In [49]:
```
def preprocess(
    doc,
    min_token_len=2,
    irrelevant_pos=["ADV", "PRON", "CCONJ", "PUNCT", "PART", "DET", "AD
):
    """
    Given text, min_token_len, and irrelevant_pos carry out preprocessi
    and return a preprocessed string.

    Parameters
    -------------
    doc : (spaCy doc object)
        the spacy doc object of the text
    min_token_len : (int)
        min_token_length required
    irrelevant_pos : (list)
        a list of irrelevant pos tags

    Returns
    -------------
    (str) the preprocessed text
    """

    clean_text = []

    for token in doc:
        if (
            token.is_stop == False  # Check if it's not a stopword
            and len(token) > min_token_len  # Check if the word meets m
            and token.pos_ not in irrelevant_pos
        ):  # Check if the POS is in the acceptable POS tags
            lemma = token.lemma_  # Take the lemma of the word
            clean_text.append(lemma.lower())
    return " ".join(clean_text)
```

In [50]:
```
wiki_df["text_pp"] = [preprocess(text) for text in nlp.pipe(wiki_df["te
```

In [51]:
```
wiki_df
```

Out[51]:

	wiki query	text	text_pp
0	Artificial Intelligence	Artificial intelligence (AI) is intelligence —p...	artificial intelligence intelligence perceive ...
1	unsupervised learning	Supervised learning (SL) is a machine learning...	supervised learning machine learn paradigm pro...
2	Supreme Court of Canada	The Supreme Court of Canada (SCC; French: Cour...	supreme court canada scc french cour suprême c...
3	Peace, Order, and Good Government	In many Commonwealth jurisdictions, the phrase...	commonwealth jurisdiction phrase peace order g...
4	Canadian constitutional law	Canadian constitutional law (French: droit con...	canadian constitutional law french droit const...
5	ice hockey	Ice hockey (or simply hockey) is a team sport ...	ice hockey hockey team sport play ice skate ic...

Training LDA with gensim (https://radimrehurek.com/gensim/models/ldamodel.html)

To train an LDA model with gensim (https://radimrehurek.com/gensim/models/ldamodel.html), you need

- Document-term matrix
- Dictionary (vocabulary)
- The number of topics (K): `num_topics`
- The number of passes: `passes`

`Gensim's doc2bow`

- Let's first create a dictionary using `corpora.Dictionary` (https://radimrehurek.com/gensim/corpora/dictionary.html).

In [52]:
```
corpus = [doc.split() for doc in wiki_df["text_pp"].tolist()]
dictionary = corpora.Dictionary(corpus)  # Create a vocabulary for the
pd.DataFrame(
    dictionary.token2id.keys(), index=dictionary.token2id.values(), col
)
```

Out[52]:

	Word
0	1.6
1	134,777
2	167,038
3	1863
4	1943
...	...
3880	wrist
3881	yale
3882	york
3883	youth
3884	zhenskaya

3885 rows × 1 columns

`Gensim's doc2bow`

- Now let's convert our corpus into document-term matrix for LDA using `dictionary.doc2bow` (https://radimrehurek.com/gensim/corpora/dictionary.html#gensim.corpora.dictionary.Dictionary.

- For each document, it stores the frequency of each token in the document in the format (token_id, frequency)

In [53]:
```
doc_term_matrix = [dictionary.doc2bow(doc) for doc in corpus]
doc_term_matrix[1][:20]
```

Out[53]:
```
[(57, 4),
 (68, 3),
 (74, 1),
 (80, 3),
 (88, 1),
 (89, 3),
 (114, 53),
 (123, 1),
 (132, 4),
 (133, 1),
 (134, 1),
 (146, 1),
 (147, 1),
 (148, 4),
 (149, 3),
 (164, 1),
 (166, 1),
 (174, 2),
 (182, 2),
 (192, 4)]
```

Now we are ready to train an LDA model.

In [54]:
```
from gensim.models import LdaModel

num_topics = 3

lda = models.LdaModel(
    corpus=doc_term_matrix,
    id2word=dictionary,
    num_topics=num_topics,
    random_state=42,
    passes=10,
)
```

Examine the topics and topic distribution for a document in our LDA model

In [55]:
```
lda.print_topics(num_words=4)  # Topics
```

Out[55]:
```
[(0, '0.002*"hockey" + 0.001*"ice" + 0.001*"game" + 0.001*"league"'),
 (1,
  '0.008*"algorithm" + 0.008*"intelligence" + 0.007*"learning" + 0.007
*"\\displaystyle"'),
 (2, '0.029*"hockey" + 0.018*"ice" + 0.014*"team" + 0.014*"player"')]
```

In [56]:
```
print("Document: ", wiki_df.iloc[0][0])
print("Topic assignment for document: ", lda[doc_term_matrix[0]])  # To
```

```
Document:  Artificial Intelligence
Topic assignment for document:  [(1, 0.9998414)]
```

Visualize topics

In [57]:
```
vis = gensimvis.prepare(lda, doc_term_matrix, dictionary, sort_topics=F
vis
```

Out[57]:

(Optional) Topic modeling with `sklearn`

- We are using `Gensim` LDA so that we'll be able to use `CoherenceModel` to evaluate topic model later.
- Bit we can also train an LDA model with `sklearn`.

In [58]:
```
from sklearn.feature_extraction.text import CountVectorizer

vec = CountVectorizer()
X = vec.fit_transform(wiki_df["text_pp"])
```

In [59]:
```
from sklearn.decomposition import LatentDirichletAllocation

n_topics = 3
lda = LatentDirichletAllocation(
    n_components=n_topics, learning_method="batch", max_iter=10, random
)
document_topics = lda.fit_transform(X)
```

In [60]:
```
print("lda.components_.shape: {}".format(lda.components_.shape))
```

```
lda.components_.shape: (3, 3814)
```

In [61]:
```
sorting = np.argsort(lda.components_, axis=1)[:, ::-1]
feature_names = np.array(vec.get_feature_names())
```

In [65]:
```
## No need to look at this code
def print_topics(topics, feature_names, sorting, topics_per_chunk=6,
                 n_words=20):
    for i in range(0, len(topics), topics_per_chunk):
        # for each chunk:
        these_topics = topics[i: i + topics_per_chunk]
        # maybe we have less than topics_per_chunk left
        len_this_chunk = len(these_topics)
        # print topic headers
        print(("topic {:<8}" * len_this_chunk).format(*these_topics))
        print(("-------- {0:<5}" * len_this_chunk).format(""))
        # print top n_words frequent words
        for i in range(n_words):
            try:
                print(("{:<14}" * len_this_chunk).format(
                    *feature_names[sorting[these_topics, i]]))
            except:
                pass
        print("\n")

print_topics(
    topics=range(3),
    feature_names=feature_names,
    sorting=sorting,
    topics_per_chunk=5,
    n_words=10,
)

```

```
topic 0       topic 1       topic 2
--------      --------      --------
hockey        court         displaystyle
ice           law           algorithm
team          intelligence  function
player        provincial    training
game          government    learn
play          power         learning
league        federal       datum
penalty       canada        feature
puck          artificial    input
canada        justice       supervised
```

Basic text preprocessing

Introduction

- Why do we need preprocessing?
  - Text data is unstructured and messy.
  - We need to "normalize" it before we do anything interesting with it.
- Example:
  - **Lemma**: Same stem, same part-of-speech, roughly the same meaning
    - Vancouver's → Vancouver
    - computers → computer
    - rising → rise, rose, rises

Tokenization

- Sentence segmentation
  - Split text into sentences
- Word tokenization
  - Split sentences into words

Tokenization: sentence segmentation

> MDS is a Master's program at UBC in British Columbia. MDS teaching team is truly multicultural!! Dr. George did his Ph.D. in Scotland. Dr. Timbers, Dr. Ostblom, Dr. Rodríguez-Arelis, and Dr. Kolhatkar did theirs in Canada. Dr. Gelbart did his PhD in the U.S.

- How many sentences are there in this text?

In [66]:

```
### Let's do sentence segmentation on "."
text = (
    "MDS is a Master's program at UBC in British Columbia. "
    "MDS teaching team is truly multicultural!! "
    "Dr. George did his Ph.D. in Scotland. "
    "Dr. Timbers, Dr. Ostblom, Dr. Rodríguez-Arelis, and Dr. Kolhatkar
    "Dr. Gelbart did his PhD in the U.S."
)

print(text.split("."))
```

```
["MDS is a Master's program at UBC in British Columbia", ' MDS teaching t
eam is truly multicultural!! Dr', ' George did his Ph', 'D', ' in Scotlan
d', ' Dr', ' Timbers, Dr', ' Ostblom, Dr', ' Rodríguez-Arelis, and Dr', '
Kolhatkar did theirs in Canada', ' Dr', ' Gelbart did his PhD in the U',
'S', '']
```

Sentence segmentation

- In English, period (.) is quite ambiguous. (In Chinese, it is unambiguous.)
  - Abbreviations like Dr., U.S., Inc.
  - Numbers like 60.44%, 0.98
- ! and ? are relatively ambiguous.
- How about writing regular expressions?
- A common way is using off-the-shelf models for sentence segmentation.

In [67]:
```
### Let's try to do sentence segmentation using nltk
from nltk.tokenize import sent_tokenize

sent_tokenized = sent_tokenize(text)
print(sent_tokenized)
```

```
["MDS is a Master's program at UBC in British Columbia.", 'MDS teaching t
eam is truly multicultural!!', 'Dr. George did his Ph.D. in Scotland.',
'Dr. Timbers, Dr. Ostblom, Dr. Rodríguez-Arelis, and Dr. Kolhatkar did th
eirs in Canada.', 'Dr. Gelbart did his PhD in the U.S.']
```

Word tokenization

> MDS is a Master's program at UBC in British Columbia.

- How many words are there in this sentence?
- Is whitespace a sufficient condition for a word boundary?

Word tokenization

> MDS is a Master's program at UBC in British Columbia.

- What's our definition of a word?
  - Should British Columbia be one word or two words?
  - Should punctuation be considered a separate word?
  - What about the punctuations in `U.S.` ?
  - What do we do with words like `Master's` ?
- This process of identifying word boundaries is referred to as **tokenization**.
- You can use regex but better to do it with off-the-shelf ML models.

In [68]:
```
### Let's do word segmentation on white spaces
print("Splitting on whitespace: ", [sent.split() for sent in sent_token

### Let's try to do word segmentation using nltk
from nltk.tokenize import word_tokenize

word_tokenized = [word_tokenize(sent) for sent in sent_tokenized]
# This is similar to the input format of word2vec algorithm
print("\n\n\nTokenized: ", word_tokenized)
```

```
Splitting on whitespace:  [['MDS', 'is', 'a', "Master's", 'program', 'a
t', 'UBC', 'in', 'British', 'Columbia.'], ['MDS', 'teaching', 'team', 'i
s', 'truly', 'multicultural!!'], ['Dr.', 'George', 'did', 'his', 'Ph.D.',
'in', 'Scotland.'], ['Dr.', 'Timbers,', 'Dr.', 'Ostblom,', 'Dr.', 'Rodríg
uez-Arelis,', 'and', 'Dr.', 'Kolhatkar', 'did', 'theirs', 'in', 'Canad
a.'], ['Dr.', 'Gelbart', 'did', 'his', 'PhD', 'in', 'the', 'U.S.']]



Tokenized:  [['MDS', 'is', 'a', 'Master', "'s", 'program', 'at', 'UBC',
'in', 'British', 'Columbia', '.'], ['MDS', 'teaching', 'team', 'is', 'tru
ly', 'multicultural', '!', '!'], ['Dr.', 'George', 'did', 'his', 'Ph.D.',
'in', 'Scotland', '.'], ['Dr.', 'Timbers', ',', 'Dr.', 'Ostblom', ',', 'D
r.', 'Rodríguez-Arelis', ',', 'and', 'Dr.', 'Kolhatkar', 'did', 'theirs',
'in', 'Canada', '.'], ['Dr.', 'Gelbart', 'did', 'his', 'PhD', 'in', 'th
e', 'U.S', '.']]
```

Word segmentation

For some languages you need much more sophisticated tokenizers.

- For languages such as Chinese, there are no spaces between words.
  - jieba (https://github.com/fxsjy/jieba) is a popular tokenizer for Chinese.
- German doesn't separate compound words.
  - Example: *rindfleischetikettierungsüberwachungsaufgabenübertragungsgesetz*
  - (the law for the delegation of monitoring beef labeling)

Types and tokens

- Usually in NLP, we talk about
  - **Type** an element in the vocabulary
  - **Token** an instance of that type in running text

Exercise for you

> UBC is located in the beautiful province of British Columbia. It's very close to the U.S. border. You'll get to the USA border in about 45 mins by car.

- Consider the example above.

- How many types? (task dependent)
- How many tokens?

Other commonly used preprocessing steps

- Punctuation and stopword removal
- Stemming and lemmatization

Punctuation and stopword removal

- The most frequently occurring words in English are not very useful in many NLP tasks.
  - Example: *the* , *is* , *a* , and punctuation
- Probably not very informative in many tasks

In [69]:
```
# Let's use `nltk.stopwords`.
# Add punctuations to the list.
stop_words = list(set(stopwords.words("english")))
import string

punctuation = string.punctuation
stop_words += list(punctuation)
# stop_words.extend(['``','`','br','"',"”", "''", "'s"])
print(stop_words)
```

```
['s', 'won', 'to', "you're", "don't", 'any', 'now', "isn't", 'up', 'thi
s', 'too', 'under', 'for', 'd', 'themselves', 'in', 'hasn', "shan't", 'be
fore', 'doesn', 'your', 'here', 'out', 'such', 'during', 'and', "were
n't", "mightn't", 'then', 'about', 'couldn', 'if', 'being', 'don', 'doe
s', 'mightn', 'shouldn', 'same', 'wasn', "you've", 'own', 'isn', 'i', 'hi
s', 'just', 'all', 'the', 'at', 'when', 'can', "it's", 'against', 'on',
'was', 'between', 'or', 'once', 'there', 'other', "aren't", 'we', 'hadn',
'from', 'himself', 'needn', 'didn', 'these', 'her', 'nor', 'shan', 'me',
'their', 'yourself', 'above', 'down', 'll', "wouldn't", "hasn't", "you'l
l", 't', 'have', 'whom', 'those', "you'd", 'that', 'only', "won't", 'beca
use', 'a', 'why', 'each', 'as', 'aren', 'theirs', "wasn't", 'them', 'ove
r', 'very', 'myself', 'were', 'been', 'o', 'doing', 'more', 'ma', 'agai
n', 'ain', 'hers', 'so', 'through', 'both', "haven't", 'than', 'y', 'ha
s', "didn't", 'm', 'itself', 'am', 'she', 'off', 'not', 'having', 'had',
"doesn't", "that'll", 've', 'him', 'an', 'yourselves', 'while', 'after',
'few', 'mustn', 'do', 'its', "couldn't", 'they', 'my', 'yours', 'but', 'i
nto', 'did', 'most', 'where', 'will', 'is', 'with', 'ourselves', 'who',
'no', "she's", 'until', 'our', "should've", 'it', 'you', "needn't", 'ho
w', 'are', 'below', 'further', "hadn't", 'wouldn', 'should', "shouldn't",
'haven', 'what', 'which', 'by', 'some', 'weren', "mustn't", 'be', 're',
'herself', 'ours', 'he', 'of', '!', '"', '#', '$', '%', '&', "'", '(',
')', '*', '+', ',', '-', '.', '/', ':', ';', '<', '=', '>', '?', '@',
'[', '\\', ']', '^', '_', '`', '{', '|', '}', '~']
```

In [70]:
```
### Get rid of stop words
preprocessed = []
for sent in word_tokenized:
    for token in sent:
        token = token.lower()
        if token not in stop_words:
            preprocessed.append(token)
print(preprocessed)
```

```
['mds', 'master', "'s", 'program', 'ubc', 'british', 'columbia', 'mds',
'teaching', 'team', 'truly', 'multicultural', 'dr.', 'george', 'ph.d.',
'scotland', 'dr.', 'timbers', 'dr.', 'ostblom', 'dr.', 'rodríguez-areli
s', 'dr.', 'kolhatkar', 'canada', 'dr.', 'gelbart', 'phd', 'u.s']
```

Lemmatization

- For many NLP tasks (e.g., web search) we want to ignore morphological differences between words
  - Example: If your search term is "studying for ML quiz" you might want to include pages containing "tips to study for an ML quiz" or "here is how I studied for my ML quiz"
- Lemmatization converts inflected forms into the base form.

In [71]:
```
import nltk

nltk.download("wordnet")
nltk.download('omw-1.4')

```

```
[nltk_data] Downloading package wordnet to /Users/mathias/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /Users/mathias/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

Out[71]:
```
True
```

In [72]:
```
# nltk has a lemmatizer
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
print("Lemma of studying: ", lemmatizer.lemmatize("studying", "v"))
print("Lemma of studied: ", lemmatizer.lemmatize("studied", "v"))
```

```
Lemma of studying:  study
Lemma of studied:  study
```

Stemming

- Has a similar purpose but it is a crude chopping of affixes
  - *automates, automatic, automation* all reduced to *automat*.
- Usually these reduced forms (stems) are not actual words themselves.
- A popular stemming algorithm for English is PorterStemmer.

- Beware that it can be aggressive sometimes.

In [73]:
```
from nltk.stem.porter import PorterStemmer

text = (
    "UBC is located in the beautiful province of British Columbia... "
    "It's very close to the U.S. border."
)
ps = PorterStemmer()
tokenized = word_tokenize(text)
stemmed = [ps.stem(token) for token in tokenized]
print("Before stemming: ", text)
print("\n\nAfter stemming: ", " ".join(stemmed))
```

```
Before stemming:  UBC is located in the beautiful province of British Col
umbia... It's very close to the U.S. border.


After stemming:  ubc is locat in the beauti provinc of british columbia
... it 's veri close to the u.s. border .
```

Other tools for preprocessing

- We used Natural Language Processing Toolkit (nltk) (https://www.nltk.org/) above
- Many available tools
- spaCy (https://spacy.io/)

spaCy (https://spacy.io/)

- Industrial strength NLP library.
- Lightweight, fast, and convenient to use.
- spaCy does many things that we did above in one line of code!
- Also has multi-lingual (https://spacy.io/models/xx) support.

In [74]:
```
import spacy

# Load the model
nlp = spacy.load("en_core_web_md")
text = (
    "MDS is a Master's program at UBC in British Columbia. "
    "MDS teaching team is truly multicultural!! "
    "Dr. George did his Ph.D. in Scotland. "
    "Dr. Timbers, Dr. Ostblom, Dr. Rodríguez-Arelis, and Dr. Kolhatkar
    "Dr. Gelbart did his PhD in the U.S."
)

doc = nlp(text)
```

In [75]:
```
# Accessing tokens
tokens = [token for token in doc]
print("\nTokens: ", tokens)

# Accessing lemma
lemmas = [token.lemma_ for token in doc]
print("\nLemmas: ", lemmas)

# Accessing pos
pos = [token.pos_ for token in doc]
print("\nPOS: ", pos)
```

```
Tokens:  [MDS, is, a, Master, 's, program, at, UBC, in, British, Columbi
a, ., MDS, teaching, team, is, truly, multicultural, !, !, Dr., George, d
id, his, Ph.D., in, Scotland, ., Dr., Timbers, ,, Dr., Ostblom, ,, Dr., R
odríguez, -, Arelis, ,, and, Dr., Kolhatkar, did, theirs, in, Canada, .,
Dr., Gelbart, did, his, PhD, in, the, U.S.]

Lemmas:  ['mds', 'be', 'a', 'Master', "'s", 'program', 'at', 'UBC', 'in',
'British', 'Columbia', '.', 'mds', 'teaching', 'team', 'be', 'truly', 'mu
lticultural', '!', '!', 'Dr.', 'George', 'do', 'his', 'ph.d.', 'in', 'Sco
tland', '.', 'Dr.', 'Timbers', ',', 'Dr.', 'Ostblom', ',', 'Dr.', 'Rodríg
uez', '-', 'Arelis', ',', 'and', 'Dr.', 'Kolhatkar', 'do', 'theirs', 'i
n', 'Canada', '.', 'Dr.', 'Gelbart', 'do', 'his', 'phd', 'in', 'the', 'U.
S.']

POS:  ['NOUN', 'AUX', 'DET', 'PROPN', 'PART', 'NOUN', 'ADP', 'PROPN', 'AD
P', 'PROPN', 'PROPN', 'PUNCT', 'NOUN', 'NOUN', 'NOUN', 'AUX', 'ADV', 'AD
J', 'PUNCT', 'PUNCT', 'PROPN', 'PROPN', 'VERB', 'PRON', 'NOUN', 'ADP', 'P
ROPN', 'PUNCT', 'PROPN', 'PROPN', 'PUNCT', 'PROPN', 'PROPN', 'PUNCT', 'PR
OPN', 'PROPN', 'PUNCT', 'PROPN', 'PUNCT', 'CCONJ', 'PROPN', 'PROPN', 'VER
B', 'PRON', 'ADP', 'PROPN', 'PUNCT', 'PROPN', 'PROPN', 'VERB', 'PRON', 'N
OUN', 'ADP', 'DET', 'PROPN']
```

Other typical NLP tasks

In order to understand text, we usually are interested in extracting information from text. Some common tasks in NLP pipeline are:

- Part of speech tagging
  - Assigning part-of-speech tags to all words in a sentence.
- Named entity recognition
  - Labelling named “real-world” objects, like persons, companies or locations.
- Coreference resolution
  - Deciding whether two strings (e.g., UBC vs University of British Columbia) refer to the same entity
- Dependency parsing
  - Representing grammatical structure of a sentence

Extracting named-entities using spaCy

In [78]:
```
from spacy import displacy
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)

doc = nlp(
    "University of British Columbia "
    "is located in the beautiful "
    "province of British Columbia."
)
displacy.render(doc, style="ent")
# Text and label of named entity span
print("Named entities:\n", [(ent.text, ent.label_) for ent in doc.ents])
print("\nORG means: ", spacy.explain("ORG"))
print("GPE means: ", spacy.explain("GPE"))
```

University of British Columbia ORG is located in the beautiful province of British Columbia GPE .

```
Named entities:
 [('University of British Columbia', 'ORG'), ('British Columbia', 'GPE')]

ORG means:  Companies, agencies, institutions, etc.
GPE means:  Countries, cities, states
```

Dependency parsing using spaCy

In [79]:
```
doc = nlp("I like cats")
displacy.render(doc, style="dep")
```

I PRON like VERB cats NOUN nsubj dobj

Many other things possible

- A powerful tool
- All my Capstone groups last year used this tool.
- You can build your own rule-based searches.
- You can also access word vectors using spaCy with bigger models. (Currently we are using `en_core_web_md` model.)